



普通高等教育“十五”国家级规划教材

微型计算机 硬件技术基础

冯博琴 主编

吴宁 陈文革 张建 编著



高等教育出版社

普通高等教育“十五”国家级规划教材

微型计算机硬件技术基础

冯博琴 主编

吴 宁 陈文革 张 建 编著

高等教育出版社

内容提要

本书是教育部普通高等教育“十五”国家级规划教材。

本书从微型计算机系统的角度出发,较为全面地介绍微型计算机系统的组成及各部分的工作原理。重点分析了 80X86 系统微处理器中具典型代表性的 8086、80386 及 Pentium 4(奔腾 4)的基本结构、工作过程和基本指令系统;阐述了计算机存储系统的组成和分类、主内存和高速缓存的工作原理及典型芯片的应用、部分联机 and 脱机外存储器的工作原理和性能以及存储器中的新技术;除此之外,还用相当的篇幅介绍微型计算机系统总线结构和输入/输出技术,包括基本输入/输出方法、典型数字量和模拟量的 I/O 接口芯片的应用等;最后,简要介绍部分常用外设的工作原理、设备驱动程序及计算机中的多媒体技术。

本书覆盖面较广,在强调基本概念的基础上,引入了大量的实例来阐明各种应用问题。力求使读者通过学习,能够对微型计算机系统有一个较为全面的了解,为进一步的微型计算机应用打下坚实的基础。

本书可作为普通高等院校非电类专业本科学生的“计算机硬件技术”课程的教材,也可作为成人高等教育的培训教材及广大科技工作者的自学参考书。

图书在版编目(CIP)数据

微型计算机硬件技术基础/冯博琴主编. —北京:高等教育出版社, 2003.8

ISBN 7-04-013298-2

I. 微… II. 冯… III. 微型计算机-硬件-高等学校-教材 IV. TP360.3

中国版本图书馆 CIP 数据核字 (2003) 第 067421 号

策划编辑 何新权 责任编辑 陈红英 封面设计 于文燕 责任印制 孔 源

出版发行 高等教育出版社

社 址 北京市西城区德外大街 4 号

邮政编码 100011

总 机 010-82028899

购书热线 010-64054588

免费咨询 800-810-0598

网 址 <http://www.hep.edu.cn>

<http://www.hep.com.cn>

经 销 新华书店北京发行所

印 刷 河北新华印刷一厂

开 本 787×1092 1/16

印 张 31

字 数 620 000

版 次 2003 年 8 月第 1 版

印 次 2003 年 8 月第 1 次印刷

定 价 35.20 元

本书如有缺页、倒页、脱页等质量问题,请到所购图书销售部门联系调换。

版权所有 侵权必究

前 言

这是一本面向非电类专业本科“微型计算机硬件技术”课程的通用教材。本书从微型计算机系统的角度出发,较为全面地介绍系统的组成及各部分的工作原理、80X86 基本指令系统、总线结构、存储器系统、输入/输出技术、常用外部设备及设备驱动程序等。

全书共 9 章。第 1 章、第 2 章为基础理论部分,介绍了微型计算机的整体概念和计算机中的常用计数制、编码及二进制数的运算,为后续内容的学习打下基础。第 3 章、第 4 章、第 6 章分别介绍主机系统中的微处理器、总线及存储器系统的结构和工作原理,书中对 CPU 的阐述以 80X86 系列微处理器中具有典型代表性的 8086、80386 及 Pentium 4(奔腾 4)为例,具有一定的新颖性。第 5 章介绍 80X86 基本指令系统和寻址方式。第 7 章、第 8 章介绍输入/输出系统,包括基本输入/输出方法及常用接口芯片。第 9 章简要叙述常用外部设备、设备驱动程序及计算机中的多媒体技术。

针对非电类专业学生的特点,全书在内容的选取及叙述上具有如下几个特点:

(1) 本书从微型计算机系统的角度出发,帮助学生建立系统的整体概念。从主机到外设详细阐述了微型计算机硬件系统各部分的结构和工作原理,包括微处理器、总线结构、存储器系统、输入/输出系统及常用外部设备和设备驱动程序,内容系统。

(2) 覆盖面广、内容新颖并具实用性是本书追求的重要目标。与当前同类教材相比,本书在以下几个方面做了一定的改进,增加了对微型计算机新技术的介绍:

① 除介绍 16 位的 8086 外,本书还用相当篇幅重点介绍 80386 以及目前最新型的 Pentium 4 的内部构成、工作模式和新增指令系统。使读者在掌握微处理器基本工作原理的基础上,也能对目前最新的 CPU 技术有一定的了解。

② 对存储器部分的描写,没有完全如通常那样直接从半导体存储器和外存储器角度介绍,而是首先从存储器系统的角度介绍两类存储器系统的特点和存储器的层次结构。同时,还加强了闪速存储器(Flash Memory)部分的描述,增加了虚拟存储器管理技术、内存条的硬件技术等较为新颖和实用的内容。

③ 在总线结构部分,加入了各种流行的系统总线标准和外部总线(如 USB 总线)的介绍。

④ 在外设部分增加了设备驱动程序的概念、DOS 及 Windows 下的设备驱动程序以及计算机中的多媒体技术等内容。

(3) 考虑到汇编语言程序设计对非电类专业学生并非必需的内容,并且也较难掌握,故

本书只简单介绍了汇编语言源程序的结构框架,去掉了同类教材均包含的汇编语言程序设计部分。

(4) 作者通过多年的教学体验,深知非电类专业学生因缺乏一定的数字电路基础,且是在进入这门课时才初次涉及计算机的内部结构,在学习本课程上有一定困难,因此书中采用了较多详实的图解、实例分析和注释,帮助读者理解和掌握所介绍的内容,使读者通过本书的学习并结合相应的上机实践,最终能够对微型计算机系统的组成和各部分的工作原理有初步的了解,为进一步的微型计算机应用打下良好的基础。

因此,本书作为课堂用教材,对学生以后的工作有一定指导作用。

本书由冯博琴教授策划并任主编,参加编写的人员有吴宁(第3章、第5章、第7章、第9章和附录部分)、陈文革(第4章和第8章)、张建(第2章和第6章),冯博琴编写第1章,并最后统稿。

本书由西安电子科技大学李伯成教授主审,他为本书提出了许多宝贵的意见和建议,在此表示衷心的感谢。

由于计算机技术的发展日新月异,新技术层出不穷,受编者水平所限,书中错误和不当之处在所难免,敬请各位读者和专家批评指正,以便再版时及时修正。

编 者

2003年5月于西安交通大学

教育部计算机基础课程系列教材出版说明

进入 21 世纪之后,我国明显地加快了建设世界教育大国的步伐,现在正向世界教育强国的目标迈进。实现这个历史性任务的最为关键指标是要有国际公认的高等教育质量,而高水平的教材是一流教育质量的重要保证。

“九五”期间,教育部工科计算机基础课程教学指导委员会曾组织了一批面向 21 世纪教材,其中有不少经典之作。本届教学指导委员会参与了国家级“十五”规划教材的组织工作。为了把计算机基础教育的优秀教材及时地推荐给广大从事计算机基础教育的教师和学生,本届教学指导委员会组织了本系列教材,它包含了部分计算机基础教育的国家级面向 21 世纪教材和国家级“十五”规划教材。希望它能起到促进和推动计算机基础教育改革的作用,使我国高校的计算机基础教育的质量再上一个台阶。

计算机基础教育改革一直在不断地深化,课程体系和教学内容趋于更加合理和科学。本系列的教材与以前出版的教材会有较大的变化,这也是我们期待的。

每一本教材都有它的应用范围,适合不同的办学层次、学科、地域和人才培养模式的教材必然有差异。本系列教材将会考虑这种差异,以满足各种层次和类型的教学所需。

列入本系列的教材,当在国内同类教材的优秀之列,我们希望作者把它打造成国家级的精品教材,要求:做到“三新”,即体系新、内容新、方法新:把每一本教材都做成既有文字教材,又有电子教材,既有教科书,又有辅助教材,成为真正意义上的“立体化”。教材的出版仅是“万里长征的第一步”,要成为精品教材,作者还必须根据读者的反映和需求不断修订原作,真正做到“与时俱进”。

“一切为了教学,一切为了读者”是我们的心愿,书中不足之处,恳望教师 and 同学们指正。

教育部高等学校非计算机专业计算机基础课程教学指导分委员会

2003.7

目 录

第 1 章 微型计算机系统概述	(1)	2.4.2 浮点数	(44)
1.1 概述	(1)	2.5 二进制编码	(45)
1.1.1 微型计算机系统	(1)	习题二	(48)
1.1.2 计算机系统的层次结构	(2)	第 3 章 微处理器	(50)
1.1.3 计算机系统的应用	(3)	3.1 微处理器的一般结构	(50)
1.2 硬件系统	(5)	3.1.1 运算器	(51)
1.2.1 硬件系统的逻辑构成	(5)	3.1.2 控制器	(52)
1.2.2 硬件系统的物理构成	(7)	3.2 8086 微处理器	(54)
1.3 软件系统	(11)	3.2.1 功能结构及其特点	(54)
1.4 微型计算机的一般工作原理	(13)	3.2.2 引出线定义及总线结构	(57)
1.4.1 程序和指令	(13)	3.2.3 工作时序	(64)
1.4.2 存储程序工作原理	(14)	3.3 8086 的寄存器组	(68)
1.4.3 微型计算机的工作过程	(15)	3.3.1 通用寄存器	(68)
1.5 计算机常用术语解释	(21)	3.3.2 段寄存器组	(69)
习题一	(22)	3.3.3 控制寄存器	(70)
第 2 章 计算机中的数制和编码	(24)	3.4 存储器组织	(71)
2.1 计算机中的数制	(24)	3.4.1 物理地址与存储器的分段	(71)
2.1.1 常用计数制	(24)	3.4.2 段寄存器的使用	(73)
2.1.2 各种数制之间的转换	(26)	3.5 80X86 微处理器	(75)
2.2 无符号二进制数的运算	(28)	3.5.1 80286 微处理器	(75)
2.2.1 二进制的算术运算	(28)	3.5.2 80386 微处理器	(77)
2.2.2 无符号数的表示范围	(30)	3.5.3 Pentium 4 微处理器	(87)
2.2.3 二进制数的逻辑运算	(31)	习题三	(102)
2.2.4 基本逻辑门及常用逻辑部件	(32)	第 4 章 总线结构	(103)
2.3 带符号二进制数的表示及运算	(36)	4.1 总线的基本概念	(103)
2.3.1 带符号数的表示方法	(36)	4.1.1 概述	(103)
2.3.2 补码数与十进制数之间的转换	(38)	4.1.2 总线的分类	(104)
2.3.3 补码的运算	(39)	4.2 总线结构的类型	(105)
2.3.4 带符号数运算时的溢出问题	(41)	4.2.1 总线的系统结构	(105)
2.4 定点数与浮点数	(43)	4.2.2 总线的层次结构	(108)
2.4.1 定点数	(43)	4.3 总线技术	(112)
		4.3.1 总线的基本功能	(112)

4.3.2 总线的数据传送	(112)	6.1.2 存储器的体系结构	(230)
4.3.3 总线的仲裁控制	(114)	6.1.3 存储器的分类	(230)
4.3.4 总线驱动及出错处理	(117)	6.1.4 存储器的主要性能指标	(231)
4.3.5 总线的性能指标	(118)	6.2 随机存储器(RAM)	(233)
4.4 常用系统总线	(119)	6.2.1 存储器的一般概念	(233)
4.4.1 系统总线标准的内容	(119)	6.2.2 静态随机存储器(SRAM)	(234)
4.4.2 ISA 和 EISA 总线	(120)	6.2.3 动态随机存储器(DRAM)	(241)
4.4.3 PCI 总线	(123)	6.3 只读存储器(ROM)	(246)
4.4.4 AGP 总线	(134)	6.3.1 掩模型只读存储器(MROM) ...	(246)
4.4.5 新型总线 PCI Express	(138)	6.3.2 一次编程型只读存储器(PROM)	(246)
4.5 外部设备总线	(140)	6.3.3 可重写只读存储器(EPROM) ...	(247)
4.5.1 通用串行总线(USB)	(141)	6.3.4 电擦除可重写只读存储器(EEPROM 或 E ² PROM)	(250)
4.5.2 IEEE 1394 总线	(157)	6.3.5 闪速存储器(Flash Memory)	(250)
习题四	(159)	6.4 微型计算机系统存储器组织 ...	(255)
第 5 章 指令系统	(161)	6.4.1 存储器的扩展技术	(255)
5.1 指令系统概述	(161)	6.4.2 CPU 与主存储器的连接	(257)
5.1.1 指令的格式	(162)	6.4.2 PC 机的存储器组织	(259)
5.1.2 指令中的操作数	(163)	6.5 高速缓存(Cache)	(262)
5.1.3 指令的字长及执行时间	(165)	6.5.1 Cache 的工作原理和基本结构	(262)
5.2 寻址方式	(169)	6.5.2 Cache 与 DRAM 的存取一致性	(265)
5.2.1 寻找操作数的寻址方式	(169)	6.5.3 Cache 的分级体系结构	(266)
5.2.2 寻找转移地址的寻址方式	(175)	6.6 存储器管理技术	(267)
5.3 8086 指令系统	(177)	6.6.1 虚拟存储器的实现机制	(267)
5.3.1 数据传送指令	(177)	6.6.2 Windows 9X 的内存管理	(272)
5.3.2 算术运算指令	(187)	6.7 新一代内存条的硬件技术发展	(274)
5.3.3 逻辑运算和移位指令	(194)	6.7.1 DRAM 的发展	(275)
5.3.4 串操作指令	(200)	6.7.2 几种内存条的封装标准	(277)
5.3.5 程序控制指令	(203)	6.7.3 内存条的规范	(278)
5.3.6 处理器控制指令	(213)	6.8 外存储器简介	(280)
5.4 80X86 新增指令及汇编语言源程序结构	(214)	6.8.1 硬盘及硬盘驱动器	(280)
5.4.1 80X86 虚地址下的寻址方式 ...	(214)	6.8.2 软盘及软盘驱动器	(283)
5.4.2 80X86 新增指令	(216)	6.8.3 光盘	(284)
5.4.3 汇编语言源程序结构	(220)	6.8.4 可移动外存储器(USB 硬盘) ...	(285)
习题五	(224)	习题六	(286)
第 6 章 存储系统	(226)		
6.1 概述	(226)		
6.1.1 存储系统概念	(226)		

第7章 输入/输出技术	(288)	8.2.3 可编程串行接口 8250	(377)
7.1 输入/输出系统概述	(288)	8.3 模拟量输入/输出接口	(389)
7.1.1 输入/输出系统的特点	(288)	8.3.1 模拟量输入/输出通道	(390)
7.1.2 输入/输出接口的基本功能 ...	(289)	8.3.2 数模(D/A)转换器	(391)
7.1.3 I/O 端口	(291)	8.3.3 模数(A/D)转换器	(403)
7.2 常用输入/输出方法	(295)	8.3.4 A/D 转换器和 D/A 转换器的 综合应用实例	(410)
7.2.1 程序控制方式	(295)	习题八	(412)
7.2.2 中断控制方式	(299)	第9章 常用外部设备及设备驱动程序 ...	(415)
7.2.3 直接存储器存取方式(DMA) ...	(300)	9.1 常用外部设备	(415)
7.2.4 I/O 通道控制方式	(302)	9.1.1 键盘	(415)
7.3 中断技术	(303)	9.1.2 鼠标	(420)
7.3.1 中断的一般概念	(303)	9.1.3 显示系统	(422)
7.3.2 中断响应的工作过程	(306)	9.1.4 打印机	(428)
7.3.3 8086/8088 中断系统	(311)	9.1.5 网卡	(432)
7.3.4 中断程序设计	(317)	9.1.6 调制解调器	(436)
7.3.5 保护模式下的中断响应	(319)	9.2 设备驱动程序	(440)
7.4 中断控制器 8259A	(321)	9.2.1 设备驱动程序的一般概念	(440)
7.4.1 8259A 的引脚及内部结构	(321)	9.2.2 Windows 9X 设备驱动程序	(441)
7.4.2 8259A 的工作原理	(324)	9.3 计算机中的多媒体技术	(447)
7.4.3 8259A 的命令字	(329)	9.3.1 多媒体计算机	(447)
7.4.4 8259A 在微型计算机系统中 的应用	(334)	9.3.2 多媒体技术概述	(450)
习题七	(338)	9.3.3 多媒体系统的数据及 文件格式	(458)
第8章 输入/输出接口	(340)	9.3.4 声卡	(462)
8.1 简单数字接口电路	(340)	9.3.5 视频获取卡	(466)
8.1.1 接口电路的基本构成	(340)	习题九	(468)
8.1.2 基本输入接口	(341)	附录	(470)
8.1.3 基本输出接口	(343)	附录 A ASCII 码表	(470)
8.1.4 具有三态输出的锁存器	(344)	附录 B 8086/8088 指令简表	(471)
8.1.5 简单接口的应用举例	(345)	附录 C 8086/8088 的中断系统	(475)
8.2 可编程数字接口芯片	(347)	附录 D 常用伪指令简表	(481)
8.2.1 可编程定时/计数器 8253	(348)		
8.2.2 可编程并行输入/输出 接口 8255	(362)		

第1章 微型计算机系统概述

本章主要介绍微型计算机系统的整体概念、微型计算机硬件系统的逻辑结构和物理组成、微型计算机的软件系统和操作系统的基本概念,简要介绍微型计算机的一般工作原理。

1.1 概 述

计算机技术是20世纪发展最快的技术之一。在从第一台电子计算机 ENIAC(Electronic Numerical Integrator And Computer)1946年问世至今的半个多世纪中,经历了电子管电路、晶体管电路、中小规模集成电路、大规模集成电路及超大规模集成电路(VLSI)五次更新换代。从单处理器系统到多处理器系统,从定点运算、顺序处理到浮点运算、并行处理,从自主存储器到共享存储器、分步存储器系统,从指令流到数据流……技术的发展,使计算机系统的性能不断提高,复杂度不断加大,价格不断降低,体积越来越小,使用也越来越方便,已成为各个领域不可缺少的工具。

但到底什么是计算机系统?它具有什么样的结构呢?

事实上,通常所说的计算机指的是计算机系统,它不仅包含了真正意义上的计算机主机,还包括了多种与主机相连的必不可少的外部设备,如键盘、鼠标、显示器等。通常在办公桌上见到的计算机叫做微型计算机系统,由于它在性能、价格等方面的优势,已越来越广泛地应用在各个领域。另外,在军事、气象、复杂的科学计算等诸多领域中,还有更高速、具更大存储容量、更强功能的计算机,这就是通常所说的大型机或巨型机。

对于计算机系统,可按照它的性能、运算速度、指令系统功能的强弱、复杂度等因素分为巨型机、大型机、中型机、小型机和微型计算机。

1.1.1 微型计算机系统

所谓微型计算机(Microcomputer)是指体积、重量、计算能力都相对比较小的一类计算机,一般供个人使用,所以通常又称为个人计算机(PC, Personal Computer)。它诞生于20世纪70年代,随着科学技术的发展,现在一台微型计算机的性能已达到当年一台大型机的性能,而其价格却只是大型机的若干分之一。由于它体积小,价格低,性能上能满足大多数用户的需要,所以得到了迅速的发展,目前已广泛应用在各行各业乃至家庭中。

同样,这里所讲的微型计算机严格地说是指微型计算机系统。它与常说的微型计算机和微处理器是3个不同的概念,是微型计算机系统从全局到局部的3个不同的层次。微处理器是一片集成了运算和控制功能的超大规模集成电路(VLSI)芯片,随着技术的发展,它的运算速度及控制功能日益强大,是整个微型计算机系统的核心;微型计算机是通常所说的主机系统,除CPU外,它还包括存储程序和数据的内存储器、用于传送信息的总线及连接外部设备的输入/输出接口等。仅有主机的计算机是没有任何实际意义的,只有在配置了基本的外部设备(如键盘、鼠标、显示器等)、并安装好可在其上运行的各种相关软件后,它才能够成为真正有用的计算机,也就是计算机系统。因此,一个完整的计算机系统不仅应包含看得见、摸得着的硬件系统,还应包含能在硬件系统上运行、从而实现各种功能的软件系统(如图1.1所示)。

这里的硬件是指组成计算机的物理实体,是看得见、摸得着的部分。对微型计算机系统来讲,硬件包括了主机箱及其内部所有元器件组成的电路和键盘、鼠标、显示器、磁盘驱动器及打印机等外部设备。而对大型计算机,硬件系统就要复杂得多,常组装在若干个大型机柜中。

软件在早先主要指的就是程序,但按照现代软件工程的理论,软件不再仅仅指程序,还包括所有相关的文档。在这本书中,将主要讨论微型计算机硬件系统的结构。

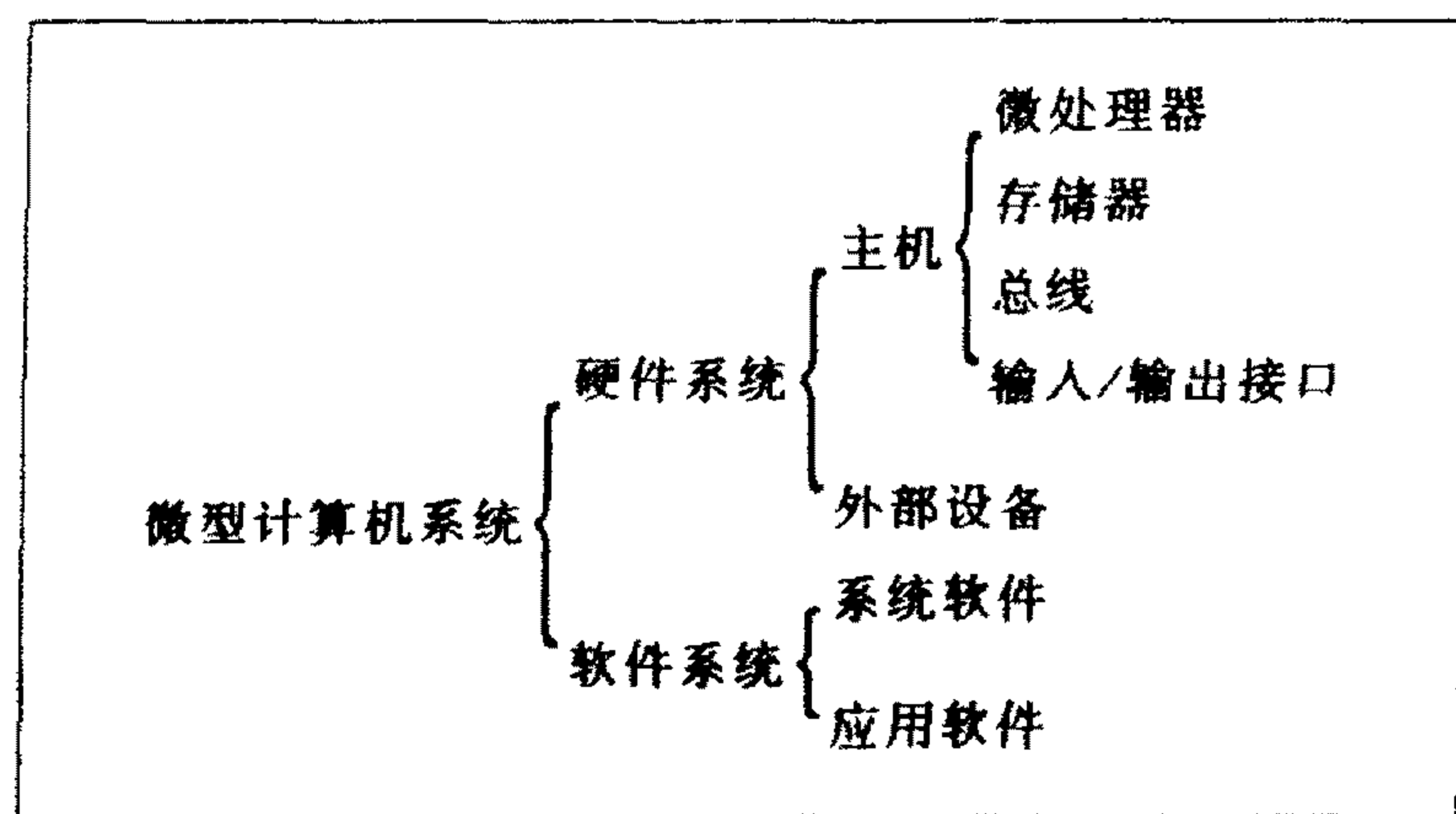


图 1.1 计算机系统的概念结构

1.1.2 计算机系统的层次结构

微型计算机系统除在概念上具有图1.1所示的结构外,还可按照功能更细地划分为图1.2所示的多级层次结构。

图中最低的两级是硬件级,是具体实现机器指定功能的中央控制部分,也是整个系统运行的物理基础。包括逻辑电路及时序电路等硬件设备以及微程序(固件),它根据各种指令操作所需的控制时序,配备一套微指令,编写出微程序,控制信息在系统内部的传送。

第三级是机器语言级。机器语言是计算机惟一能够直接识别的语言。程序员用机器语



图 1.2 计算机系统的层次结构

言编写的程序由微程序进行解释。

从第 4 级向上,一般讲属于软件系统的讨论范畴。操作系统是用来管理整个计算机系统硬件并支持用户开发应用的一种系统软件,它是运行在机器语言级上的解释程序。系统应用程序是直接为用户开发应用软件提供的工具和平台,它包括各种编译系统、网络系统及为应用程序提供开发平台的各种工具软件。最上一层才是用户级,用户可在各类系统软件的支持下完成自己的应用程序设计。另外,非计算机专业人员也能够利用这一级提供的各种应用语言,通过键盘或其他方式向计算机发出请求,进入相应的信息处理系统。

1.1.3 计算机系统的应用

随着计算机技术的飞速发展,计算机已进入了各个领域,成为现代生活不可缺少的工具。计算机除微型计算机外,还有功能更加强大、运算速度更快、相应价格也更高的小型机、中型机和巨型机。另外,除能够适用于大多数场合的通用型计算机外,还有为特殊应用设计的专用机。下面简要介绍计算机应用的几个主要方面。

1. 科学计算

科学计算一直是计算机的重要应用领域之一。如在核物理学、天文学、量子化学、石油勘探、工程设计及军事领域等,都需要依靠计算机来完成复杂的计算。事实上,发明计算机最初的动力就是因为当时的美国因人口的迅速膨胀而无法在短期内实现对人口的统计。计算机在科学计算上的特点是能够快速完成大数据量、数值变化范围宽的计算,这点对那些实时性要求高的系统尤为重要。如控制导弹的飞行,要求不断测量导弹的参数,通过计算后能

够及时做出反应,修改导弹飞行的轨迹。这类计算如果靠手工实现是很难想像的,也是不可能实现的。

2. 数据处理

数据处理是目前大多数计算机的主要工作。如企业中的经营管理、财务管理,银行系统中对储户的存取款管理、信用卡管理,大型图书馆对读者借阅信息的管理等。数据处理系统一般具有输入和输出数据量较大但计算比较简单的特点。对数据量不大的场合,通常用一台或几台单机就可以满足要求。而对大型的数据处理系统,则往往需要系统中的各计算机联网使用,例如在银行系统中,为了实现各储蓄所之间的通存通兑及在商店等消费场所使用信用卡,就需要将各地的计算机与服务器以网络的形式连接在一起,以实现信息的共享。

3. 计算机控制

对生产过程的自动控制也是计算机应用的一个很重要的方面。如大型锅炉的炉温、配料控制、化工厂的过程控制等。计算机从工业现场采集信息,经过一定的计算和处理,再送出数据,以驱动执行机构进行相应的控制。工业现场的许多信息往往是非电的物理量或模拟信号,此时还需首先将它们通过一定的处理转换为计算机能够识别的数字信号。有关这方面的内容将在第8章加以介绍。

用于工业控制的计算机,主要要求有较高的可靠性,对计算速度要求不是太高。

4. 人工智能

人工智能就是用计算机来模拟人类的思维和行动,是将人类大脑的思维过程、规则及所采取的策略等编成计算机程序,这些程序不同于一般的计算机应用程序,而是希望计算机能根据其内部存储的一些公理和推理规则,去自动探索解决问题的方法。如智能机器人,就是人工智能各种研究课题的综合产物,具有感知和理解周围环境、从而完成某种动作的能力,还能进行推理并使用简单的工具。

专家系统也属于人工智能研究的范畴,它实际上是计算机模拟专家行为的程序,如医学界建立的模拟某位老专家的综合诊断系统等。

5. 计算机辅助设计/制造

因计算机具有快速的数值计算以及较强的数据处理和模拟能力,目前在机械制造、VLSI设计等众多行业中已越来越多地利用计算机进行辅助设计和制造。采用计算机辅助设计/制造(CAD/CAM),一是解决了某些因工序过于复杂而人工难以解决的问题,如超大规模集成电路的设计和生;二是通过使设计实现自动化或半自动化,减轻了人的劳动强度并提高了设计质量。设计人员可借助CAD/CAM专用软件和输入/输出设备将设计要求或方案输入计算机,通过相应的应用程序进行计算处理后显示出结果,如果不满意还可通过鼠标、键盘等进行修改。

1.2 硬件系统

从图 1.1 已经知道,微型计算机的硬件系统主要由主机和外部设备两大部分组成,其中主机又包括了 4 个部分,即微处理器、存储器、总线和输入/输出接口。

1.2.1 硬件系统的逻辑构成

1. 微处理器(Micro Processor)

微处理器也叫做中央处理单元(Central Processing Unit, CPU),其内部包括运算器、控制器以及寄存器组,是整个硬件系统的核心。它通过专门的 CPU 插座安置在主板上。目前市场上大多数微型计算机的 CPU 都是美国 Intel 公司生产的,其系列产品由早期的 8088/8086 到现在最新型的 Pentium 4,在性能上、功能上都有大幅度的提高和改进,但其基本体系结构没有改变,且指令系统保持向下兼容。

2. 存储器(Memory)

顾名思义,存储器是计算机系统的一种记忆设备,用来存放指令、数据、运算结果以及各种需要保存的信息。是计算机系统中不可缺少的一个重要组成部分。在现代计算机系统中,通常有多种用途不同的存储器,如用于在运行中暂时存储 CPU 正在执行的指令和数据的主存储器(或称内存)、为提高系统整体存取速度而设置的高速缓冲存储器(Cache)、用于大容量信息保存的磁盘存储器和光盘存储器等,它们共同构成了计算机的存储系统。

微型计算机中的存储系统一般分为两种,一种是由主存和高速缓存(Cache)构成的 Cache 存储系统,另一种是由主存储器和磁盘存储器构成的虚拟存储系统。前者的主要目标是提高存储器的速度,而后者则主要是为了增加存储器的存储容量。(关于存储系统进一步的内容参见第 6 章。)

3. 总线(Bus)

微型计算机系统采用总线结构。所谓总线就是一组信号线的集合,是计算机系统中各部件之间传输地址、数据和控制信息的公共通路。从物理结构来看,它由一组导线和相关的控制、驱动电路组成。目前在微型计算机系统中常把总线作为一个独立部件来看待。

总线一般分为 3 个层次:

第一层为微处理器级总线,也叫做 CPU 总线,包括地址总线(Address Bus, AB)、数据总线(Data Bus, DB)和控制总线(Control Bus, CB),从 CPU 引脚上引出,用来实现 CPU 与外围控制芯片之间的连接

第二层为系统级总线,也称为 I/O 通道总线,同样包括地址总线(AB)、数据总线(DB)和控制总线(CB),用于 CPU 与存储器及扩充插槽上的各扩充板卡相连接。系统总线有多种标准,以适用于各种系统。

最后一层为外设总线,是指计算机主机与外部设备接口的总线,实际上是一种外设的接口标准。当前在微型计算机上常用的接口标准有:IDE(Integrated Drive Electronics,集成驱动器电子标准)、EIDE(Enhanced Integrated Drive Electronics,增强型集成驱动器电子标准)、SCSI(Small Computer System Interface,小型计算机系统接口)、USB(Universal Serial Bus,通用串行总线)和 IEEE 1394 五种。前两种主要是与硬盘、光驱等 IDE 设备接口,后三种是新型外部总线,可以用来连接多种外部设备。

4. 输入/输出接口(Input/Output Interface)

主机与外部设备之间的信息交换是通过输入/输出接口来进行的。例如,需要计算机处理的信息要通过键盘、鼠标、扫描仪等设备输入,而处理的结果则要通过显示器、打印机、绘图仪等设备输出,以供人们查看。这些设备与主机之间因各种原因(工作速度、信息格式、信息类型、电平等不匹配)而不能直接实现信息的传递,必须通过一个中间环节,即输入/输出接口来连接。输入/输出接口简称 I/O 接口,由图 1.3 可以看出,接口在这里起着主机与外部设备之间数据通信的“桥梁”的作用。可以试想,如果一台计算机没有 I/O 接口,它也就不具备输入/输出信息的能力,这样的计算机还有意义吗?

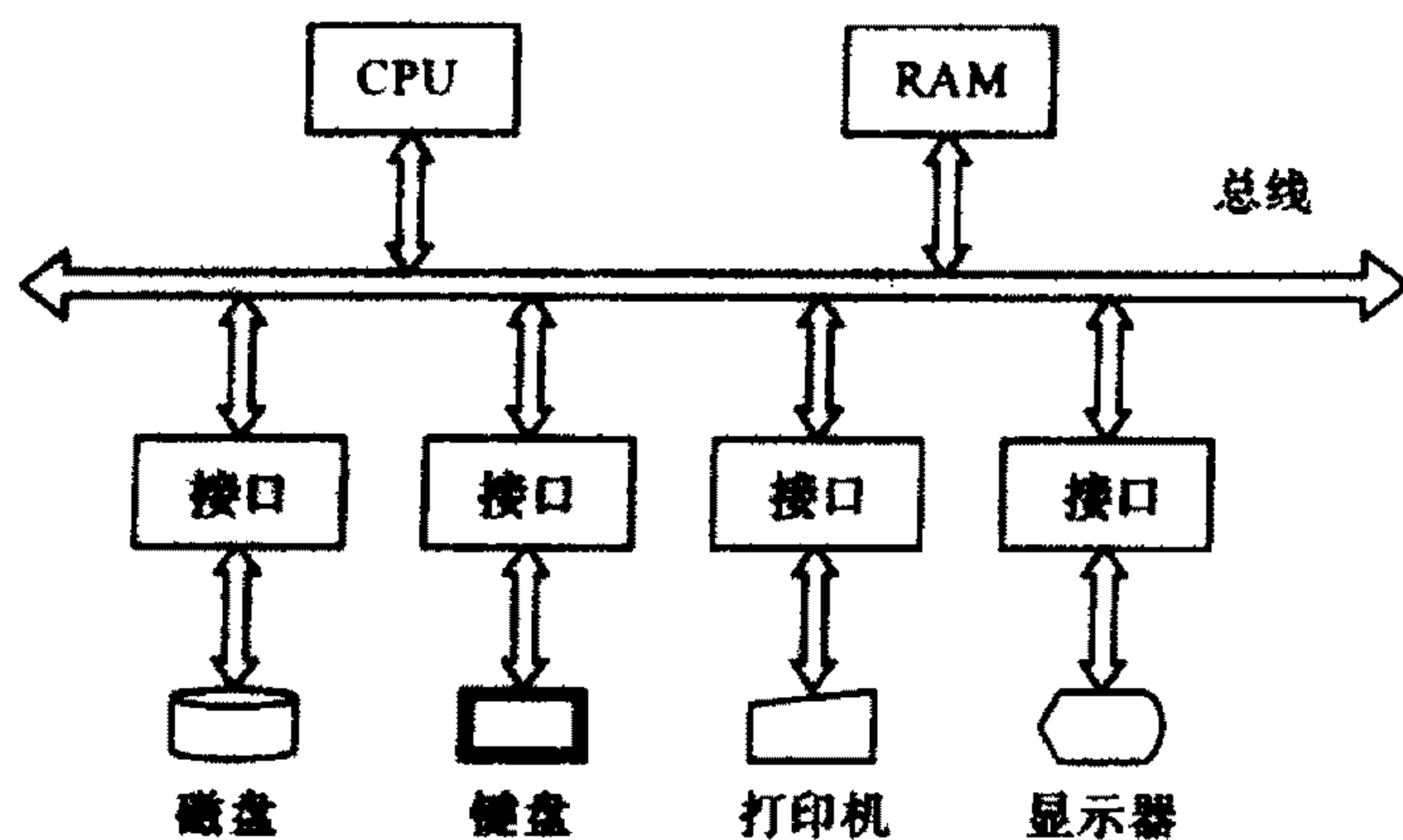


图 1.3 计算机中的接口连接示意图

现代计算机所配备的外部设备是多种多样的,可分为异步传输设备(如键盘、鼠标、调制解调器等)、同步传输设备(语音处理设备)和数据块传输设备(扫描仪、数字相机、图形输出设备等)。它们一般都配备有专门的 I/O 接口电路。这些 I/O 接口通常都做成 I/O 接口(控制)卡,插在主机板的 I/O 扩展槽上,有的也直接做在主板上。

1.2.2 硬件系统的物理构成

主机部分在物理上是安装在机箱内的主机板上,外部设备通过输入/输出接口及系统总线与主机板相连。可以用图 1.4 所示的框图来形象地表示硬件系统的组成。虚线框内的部分就称为主机板,是整个微型计算机系统的核心。

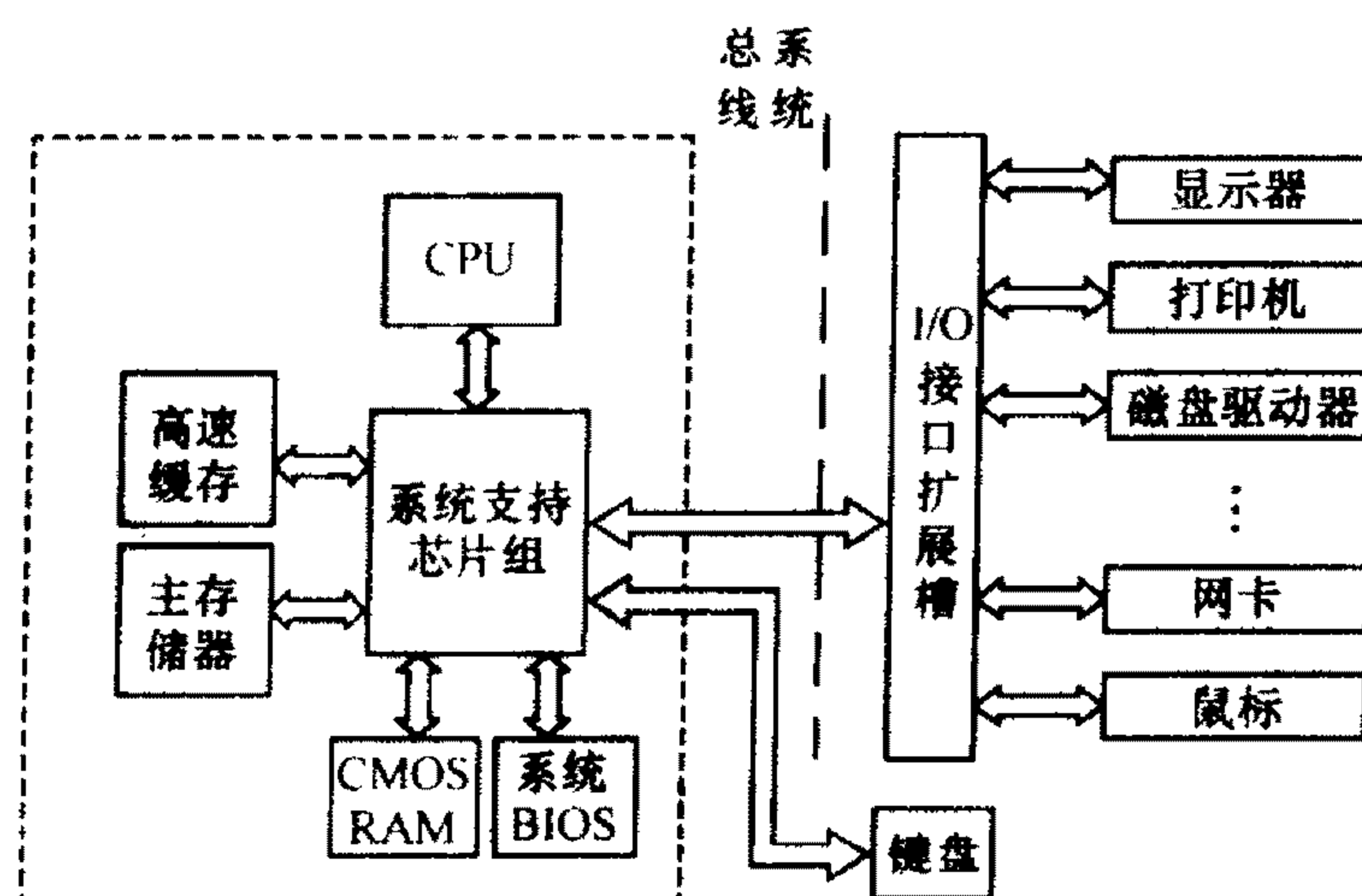


图 1.4 微型计算机的硬件系统结构框图

主机板简称主板,它几乎集中了系统的全部功能,控制着系统各部分之间的指令流和数据流,能够根据系统进程和线程的需要,有机地调度各个子系统并为实现系统的科学管理提供充分的硬件保证。

在主机板上通常有 CPU(中央处理器)、系统支持芯片组、内存芯片、I/O 接口、总线扩展槽、键盘或鼠标接口、软盘接口、IDE 接口(可接硬盘和光驱)、可充电电池以及各种开关和跳线等。最新型的一体化主板甚至还集成了显示卡、声效卡、网络卡、调制解调卡等接口部件,使用户不用再购买这类插卡就可以组成一台多媒体个人计算机。

主机板在结构上主要有 AT 主板、ATX 主板及 NLX 主板等。它们之间的区别主要在于各部件在主板上的位置排列、电源的接口外形及控制方式不同,另外在尺寸上也可能稍有不同,但不论何种结构,基本的外设接口(键盘、鼠标、串口、并口等)和总线插槽在主板上的相对位置是固定不变的。图 1.5 和图 1.6 所示分别为一个实际的 AT 主板的布局结构及 NLX 主板的外形图。

具体地讲,一个微型计算机系统的主机板主要包括以下几个部分:

- ① 微处理器(CPU);
- ② 芯片组;
- ③ 内存插槽 该插槽数量和类型影响到主存的扩展能力及扩展方式;

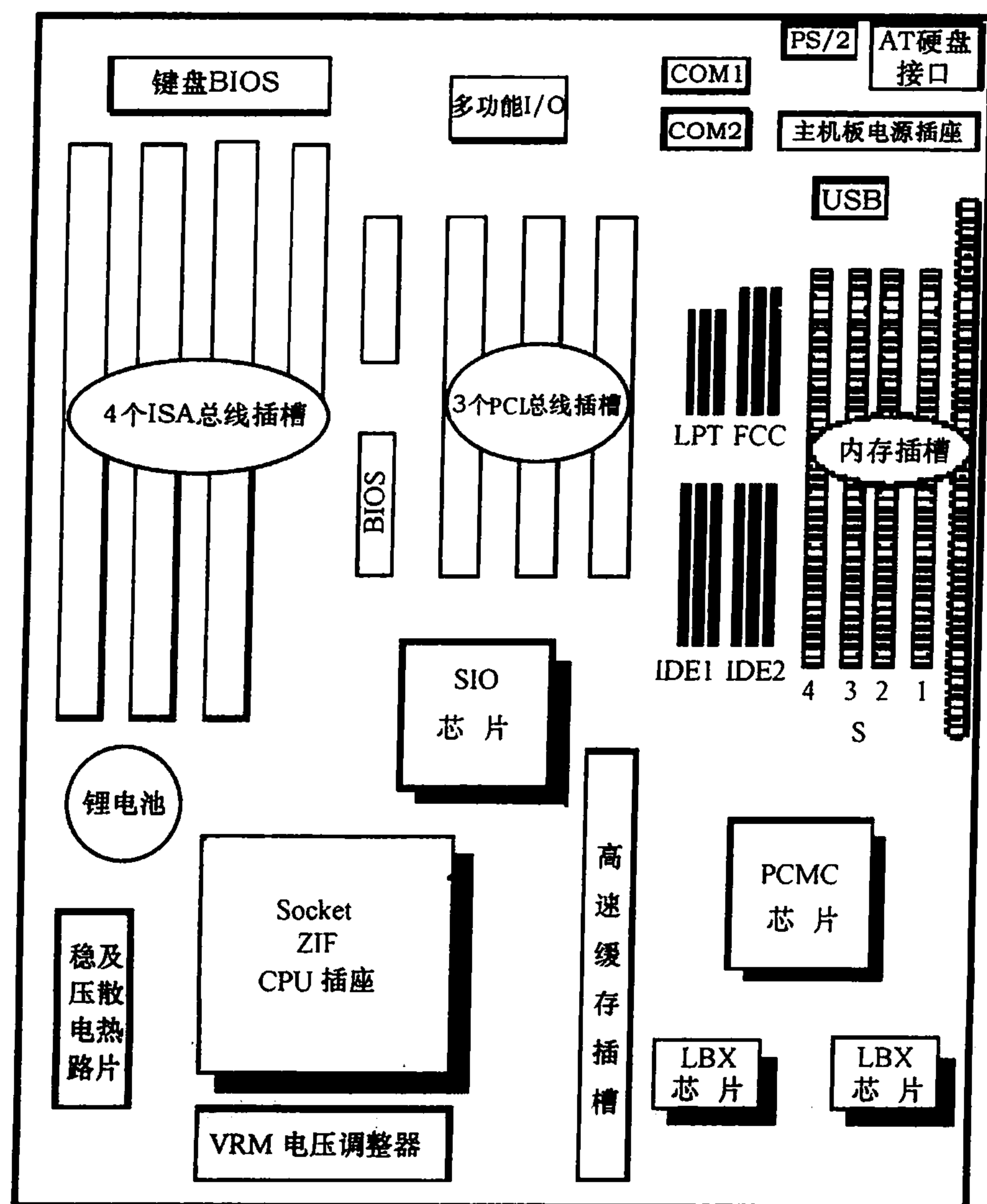


图 1.5 AT 主板的布局结构

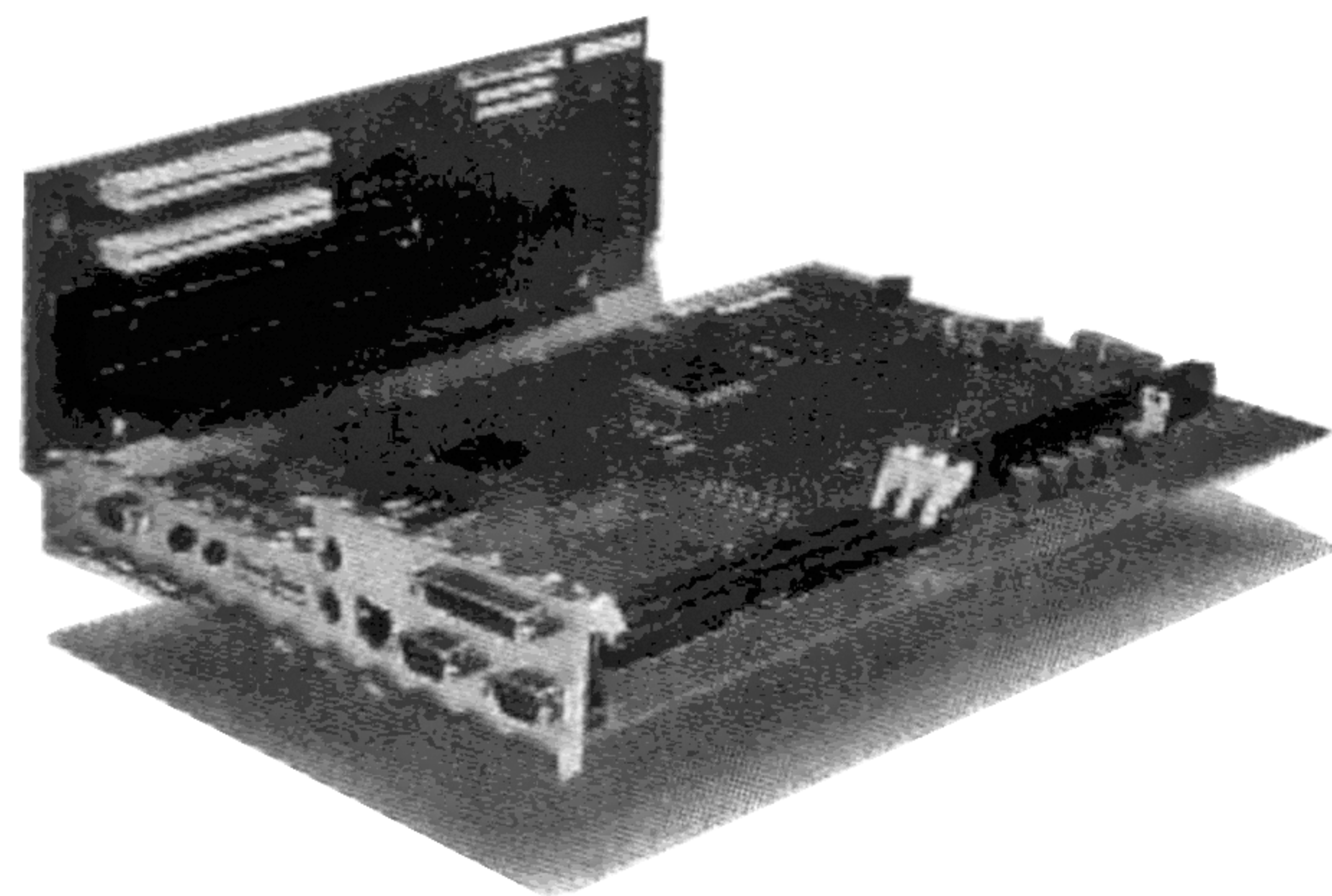


图 1.6 NLX 主板的外形图

- ④ 高速缓存;
- ⑤ 系统 BIOS(Basic Input Output System,基本输入/输出系统);
- ⑥ CMOS;
- ⑦ 总线扩展槽;
- ⑧ 串行和并行接口。

除此之外,还包括逻辑部件和跳线开关等。下面简要介绍主机板上的几个主要部件。

1. CPU 及 CPU 插座

微处理器(CPU)是整个计算机系统的核心,它安置在主机板上专门的 CPU 插座上。由于集成化程度和制造工艺的不断提高,越来越多的功能被集成到 CPU 中去,使 CPU 引脚数量不断增加,因此插座相对也越来越大。在安装形式上主要分为 Socket 和 Slot 两大类工业标准。目前的 Pentium 4 微处理器采用的是 Socket 478 插座。

2. 芯片组

芯片组是固定在主板上的一组超大规模集成电路芯片的总称,是主板的关键部件,用于控制和协调整个计算机系统的运行。系统的芯片组一旦确定,整个系统的定型和选件变化范围也就随之确定。即芯片组决定了计算机系统中各个部件的选型,它不能像 CPU、内存等其他部件那样进行简单的升级。常见的芯片组有 Intel 845PE、VIA KT400、NVIDIA nForce2 等。

3. 内存插槽

在现代微型计算机的主机板上,都安装有若干的内存插槽,只要插入相应的内存条,就可方便地构成所需容量的内存存储器。

主板上内存插槽的数量和类型对系统主存的扩展能力及扩展方式有一定影响。其插槽的线数常见的有 30 线、72 线、168 线、184 线等。目前主板上大多采用 184 线插槽。

4. 高速缓存

高速缓存也称 Cache,是为解决 CPU 与主存储器间的数据传输速率差异而设计的。它通过高速总线与 CPU 连接,容量越大对计算机总体性能提高的影响也越大。目前的微型计算机中的 Cache 一般分为两级,一级 Cache 内嵌在 CPU 中,最大容量为 64 KB;二级 Cache 固化在主板上,容量目前已达 512 KB,并且 Intel 在其 Pentium II/III 中又将二级高速缓存与 CPU 捆绑在一起,容量范围为 512 KB~2 MB(可选)。

5. CMOS

在 Intel 80286 及以后的微型计算机主板上,都有一片 CMOS RAM(简称 CMOS)的集成电路芯片,由电池供电,在计算机关机时信息不会丢失。CMOS 芯片用来存储系统运行所必需

的配置信息,如系统的存储器、显示器、磁盘驱动器等参数。对新买的微型计算机,一般要进行的设置(Setup)就是向 CMOS 中置入信息。COMS RAM 的另外一个功能是计数和提供实时的日历和时间,包括年、月、日及时、分、秒等。

6. 系统 BIOS

系统 BIOS 实际上是一组被固化存储在只读存储器 E²PROM 中的软件,所以 E²PROM 和被固化的 BIOS 合称为固件。系统 BIOS 程序包含以下几个模块:

1) 上电自检(Power - On Self Test, POST)

在微型计算机加电后,CPU 从地址为 0FFFF0H 处读取和执行指令,进入加电自检程序,测试整个微型计算机系统是否正常工作。

2) 系统初始化

包括集成电路芯片的初始化;设置中断向量表,并设置 BIOS 中包含的中断服务程序的中断向量;通过 BIOS 中的自举程序,将 DOS 操作系统中的初始引导程序装入内存,从而启动 DOS。

3) 系统设置(Setup)

装入或更新 CMOS RAM 保存的信息。在系统加电后尚未进入操作系统时,按 Del 键(或其他热键)可进入 Setup 程序,此时可修改各种可变参数或选择默认参数。

7. 总线扩展槽

主板上的扩展插槽是 CPU 通过系统总线与外部设备联系的通道,系统的各种扩展接口卡都插在扩展插槽上,如显示接口卡、声卡、解压卡、调制解调器、传真卡等。总线扩展槽类型有 ISA (Industrial Standard Architecture, 工业标准结构)、EISA (Enhanced Industrial Standard Architecture, 增强型工业标准结构)、VESA (Video Electronics Standard Association, 视频电子标准协会)和 PCI (Peripheral Connection Interface, 外围设备连接接口)等。它的发展使总线位数越来越宽,从 8 位到 64 位;传输速率也越来越快,从 16 MB/s 到 533 MB/s。PCI 总线支持即插即用的功能,减轻了板卡的配置工作。如今主板上主要预留 ISA 和 PCI 两种形式的扩展槽。另外,现在又有了一种新型的专用显示扩展接口 AGP,它具有比 PCI 更高的传输速率,但要与 AGP 接口的显示卡配合使用。

8. 串行和并行接口

在微型计算机的主机板上都配置有串行和并行接口插座,包括 RS - 232 串行口插座、USB 插座及标准并行口插座(EPP 或 ECP 规范)。

除了以上描述的这些主要部件外,主板上还包含用于连接硬盘、光驱和软驱的电缆插座(标准有 IDE、EIDE、SCSI 等)及许多不可缺少的逻辑部件和跳线开关等。所有这些部件紧密联系、相互沟通,构成了整个微型计算机数据间的交流。

1.3 软件系统

按照现代软件工程的观点,软件不仅仅指程序,还包括所有相关的文档。在这里,姑且将软件只看做程序。一个完整的计算机系统不仅包括硬件系统,还必须包括软件系统。一个没有任何软件支持的计算机是没有任何用处的。

计算机的软件可分为两大类:系统软件和应用软件。系统软件包括操作系统和系统应用程序,如网络系统、编译系统及各种工具软件。软件的核心是系统软件,而系统软件的核心是操作系统。

操作系统(Operating System, OS)是配置在计算机硬件上的第一层软件,是其他软件运行的基础。其主要功能是管理计算机系统中的各种硬件和软件资源,并为用户提供与计算机硬件系统之间的接口。在计算机上运行的其他所有的系统软件(如汇编程序、编译程序、数据库管理系统等)及各种应用程序,都要依赖于操作系统的支持。因此,操作系统在计算机系统中占据着极其重要的位置,成为无论是大型机还是微型计算机都必须配置的软件。

1. 操作系统的功能

操作系统主要具有以下 4 个方面的功能:

1) 存储器管理

存储器管理的任务是提高存储器的利用率,并在逻辑上扩充内存,为程序运行提供良好的环境,包括:

- ① 合理地分配内存空间,以提高存储器的利用率;
- ② 提供地址映射功能,将程序中的逻辑地址与内存中的物理地址相对应;
- ③ 在多道程序运行的环境下,确保每道用户程序都能在自己的内存空间中运行,而不允许访问操作系统的程序、数据和非共享的其他用户程序;
- ④ 内存扩充。借助虚拟存储技术,将容量有限的物理内存在逻辑上扩充,使用户感觉到的内存比实际物理内存要大很多,从而满足更多程序的并行运行。有关虚拟存储技术将在第 6 章介绍。

2) 进程管理

处理器管理的主要任务是对处理器进行分配和运行管理。在多道程序运行的环境下,处理器中的作业运行是以进程为单位的。所以处理器管理也就是进程的管理,包括进程控制、进程同步、进程通信和进程调度。

3) 设备管理

主要任务是根据用户要求合理分配输入/输出设备,提高 CPU 和 I/O 设备的利用率。

设备管理具有缓冲管理(管理各类缓冲区)、设备分配、设备处理(实现 CPU 与设备控制器之间的通信)及虚拟设备等功能。这里所谓的虚拟设备是指操作系统能将一个仅允许单个进程使用的物理设备变换为多个对应的逻辑设备,使一个物理设备能够为多个用户共享,从而提高了设备的利用率。

4) 文件管理

在现代计算机系统中,所有的程序和数据都是以文件的形式存放在存储器中的。操作系统中文件管理的主要任务是文件存储空间的管理(为每个文件分配合适的外存空间,并在必要时对空间进行回收)、文件目录管理(为文件建立目录项,以便按名存取及文件的共享)、文件读/写管理和文件的存取管理。

除以上几项功能外,操作系统还具有向用户提供“用户接口”的功能,包括命令接口(通过键盘发出命令控制作业运行)、程序接口(为用户程序在执行中访问系统资源而设置的一组功能子程序或库函数,可直接调用)以及现在最常见的图形接口,如 Windows 操作系统。

2. 操作系统的发展

最早的操作系统出现在上世纪 50 年代中期。在操作系统出现之前,用户要想使用计算机来完成某项任务,必须用机器语言编写程序,因为计算机的硬件只能识别 0 和 1 这样的二进制代码。所以要求用户要对计算机的硬件构成非常熟悉。另外,当一个用户使用计算机时,其他用户不能上机。只有等一个程序执行完毕取走结果后,才能让下一个用户使用,因此计算机的各类资源不能充分利用。

为了提高系统的运行效率,人们首先想到的是将零散的单一任务处理变为集中式批处理,使计算机有了相对较长的连续运行时间。为满足批处理的需要,产生了批处理控制管理程序,称为单道批处理管理程序。这就是第一代操作系统。它为计算机用户提供了一套控制命令,操作员使用这些命令向计算机表达对程序进行编译执行的要求,之后计算机就可根据命令完成一系列操作,在一个程序任务处理完后,会自动启动下一个待处理的程序。

第二代操作系统能处理多道程序,即一次可将多道程序调入内存,由操作系统根据资源的占用情况及当前程序的执行状态,统一调度这些程序,实现系统资源的共享。

在 20 世纪 70 年代中期,随着硬件技术的发展,出现了具有计算机资源综合管理功能、设计更加完善的第三代操作系统。

当前在微型计算机领域中较为流行的操作系统有 Windows、Linux 等,在中、小型计算机及服务器上的主流操作系统有 UNIX、Windows NT 等。

1.4 微型计算机的一般工作原理

冯·诺依曼和他的同事们在 1946 年提出了一个完整的现代计算机的结构雏型(如图 1.7 所示),它由 5 个部分组成,即运算器、控制器、存储器、输入设备和输出设备。运算器负责指令的执行;控制器的作用是协调并控制计算机的各个部件按程序中排好的指令序列执行;存储器是具有记忆功能的器件,用于存放程序和需要用到的数据及运算结果;而输入/输出设备则是负责从外部设备输入程序和数据,并将运算的结果送出。

冯·诺依曼型计算机是以存储程序原理为基础的。那么,什么是“存储程序”工作原理呢?在了解这一点之前先要明确什么是程序和指令。

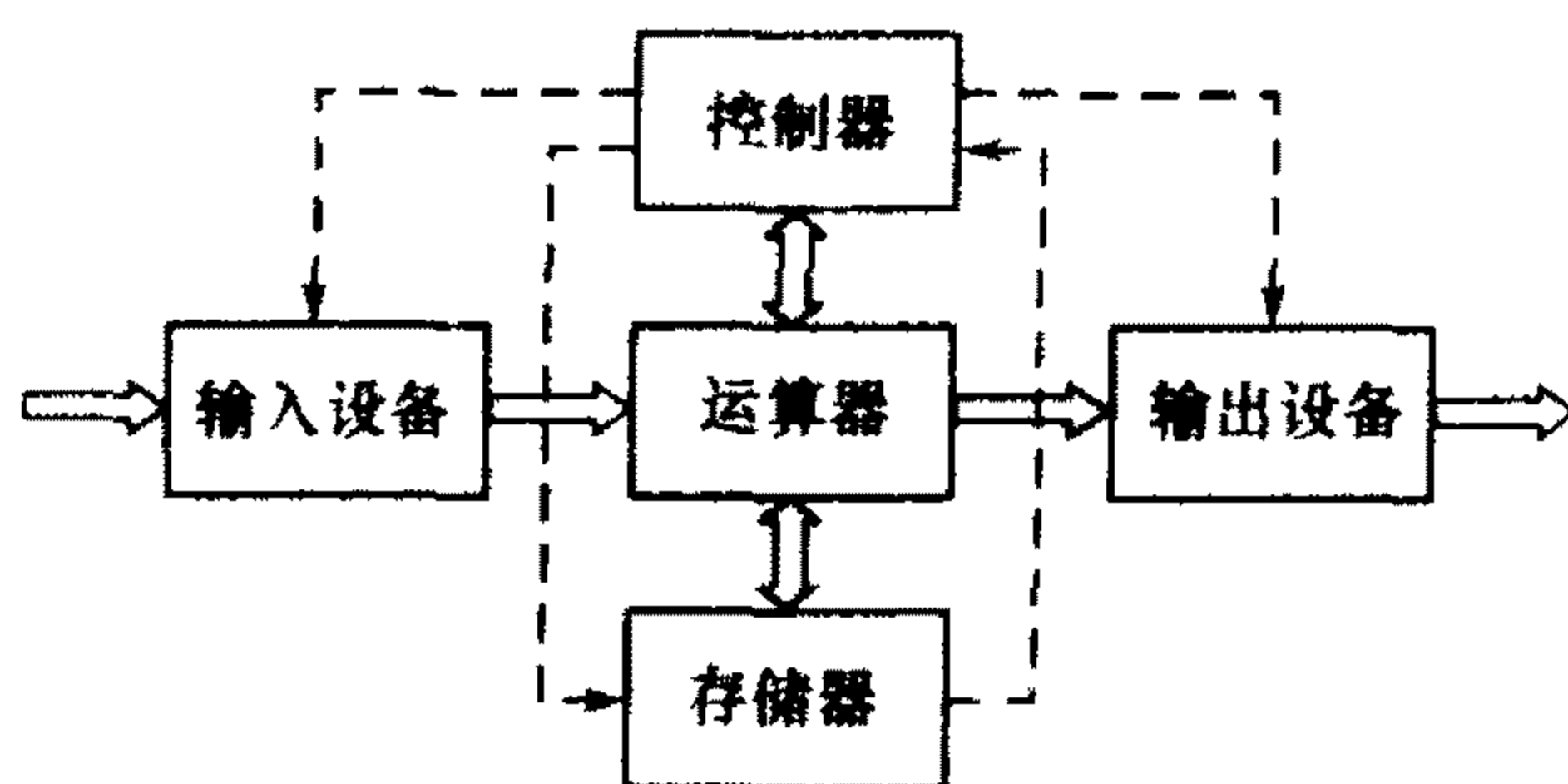


图 1.7 冯·诺依曼计算机结构示意图

1.4.1 程序和指令

计算机之所以能在当今社会各个领域扮演不可缺少的重要角色,是由于人通过编制的程序赋予了它“聪明和才干”。那么什么是程序呢?所谓程序,就是为实现某项既定的任务而向计算机发出的一组有一定顺序的基本操作命令的集合。这些基本操作命令就称为指令,每一条指令都代表计算机执行的一种基本操作,计算机的硬件系统保证了对这些指令的识别能力。当要用计算机完成某项工作时,例如,要解一道数学题时,先要把题目的解算步骤按照一定的顺序用计算机能识别并执行的基本操作命令书写出来,每一条基本操作命令都是一条机器指令,命令计算机执行规定的操作。这些指令的序列就组成了程序。

因此,程序是实现既定任务的指令序列,其中的每条指令都规定了计算机执行的一种基本操作,机器按程序安排的顺序执行指令,就可完成解题任务。其过程如图 1.8 所示。

机器指令必须满足两个条件:一是机器指令的形式是计算机能够理解的,即只能是“0”和“1”这样的二进制编码形式;二是机器指令规定的操作必须是计算机能执行的,即每条机

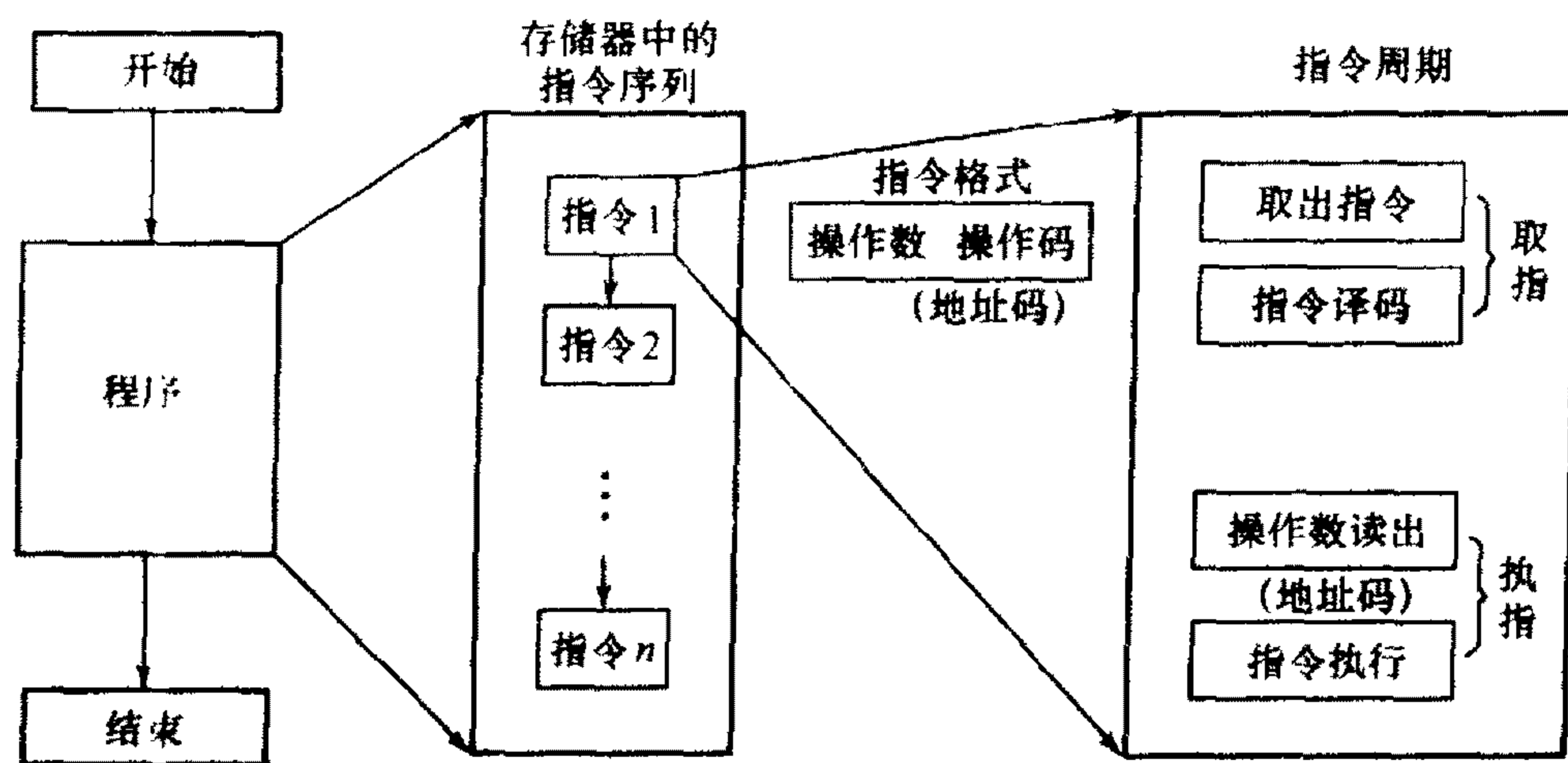


图 1.8 程序执行过程示意图

器指令的操作有相应的电子线路实现,否则用这种指令编写的程序无法在机器中实现。每台计算机的指令都有自己的格式和具体的含义,但必须指明操作性质(如加、减、乘、除、比较大小等)和参加操作的有关信息(如数据或数据存放的地址等)。

每台计算机都拥有各种类型的机器指令,机器指令的集合称为指令系统。指令系统决定了计算机的能力,也影响着计算机的结构。指令的不同组合方式,可以构成完成不同任务的程序,一台机器的指令种类是有限的,但在人们的精心设计下,实现信息处理任务的程序可以无限多,计算机严格忠实地按照程序安排的指令顺序,有条不紊地执行规定的操作,完成预定任务。

1.4.2 存储程序工作原理

存储程序原理的基本点就是指令驱动,即把计算过程描述为由许多条命令按一定顺序组成的程序,然后把程序和所需的数据一起输入计算机存储器中保存起来。当机器启动时,根据内部指令指针给出的程序第一条指令的地址,按照程序指定的逻辑顺序从存储器中一条条地读取指令、分析指令、执行指令并传送结果,自动连续地完成程序所描述的全部工作。当然,这里所说的程序必须是机器能够识别的二进制码(或者必须通过编译系统“翻译”成二进制码),它们能够和数据一样进行存取;另外,程序中的指令必须属于执行该程序的 CPU 的指令系统。

总的来讲,冯·诺依曼计算机的主要特点是以运算器和控制器为中心,输入/输出设备与存储器之间的数据传送都要经过运算器。运算器、存储器、输入/输出设备的操作及它们之间的联系由控制器集中控制。控制器通过指令流的串行驱动实现程序控制。

几十年来,计算机技术有了飞速的发展,这种以运算器为中心的原始的冯·诺依曼计算

机已逐渐演变为以存储器为中心的结构。同时,科学的进步还使计算机领域出现了许多新的技术,在系统结构方面也发生了一些重大的改进,如从基于串行算法变为适应并行算法,从而出现了向量计算机、并行计算机、多处理机等;在结构上从传统的指令驱动型变为数据驱动型和需求驱动型,出现了数据流机和归约机,等等。但从本质上讲,存储程序控制仍是现代计算机的结构基础。在计算机的结构原理上,特别是对微型计算机来讲,占主流地位的仍然是冯·诺依曼型计算机。

1.4.3 微型计算机的工作过程

微型计算机的工作过程就是执行程序的过程,而程序又是由指令序列组成的,因此执行程序的过程就是逐条地执行指令。计算机每执行一条指令,都包含着两个基本的步骤,即取指令和执行指令。

程序在运行前要先由输入设备及操作系统调入到内存储器中,当机器进入运行状态后,首先将第一条指令在内存中的地址赋给程序计数器 PC(或指令指针 IP),之后就进入取指令阶段,CPU 按照指定的地址读内存,此时读出的必为指令,此指令经过译码后,由控制器发出相应的控制信息,使运算器按照指令规定的操作去执行。一条指令执行完后,又开始取下一条指令,重复上述过程,直到遇到暂停指令或某种使程序执行暂停的意外情况才会结束。程序执行的一般过程可以简单地用图 1.9 所示的过程示意图来表示。

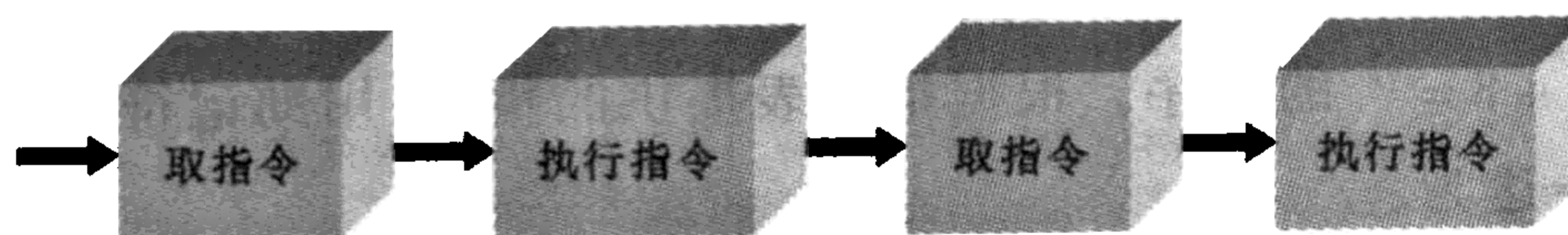


图 1.9 程序执行过程示意

为了进一步说明微型计算机的工作过程,来具体讨论一个模型机怎样执行一段简单的程序。如计算 $5 + 8 = ?$ 。这虽然是一个很简单的加法运算,但是计算机无法理解。必须要人通过编写程序,才能使计算机来完成这项工作。

要使计算机完成一项指定的工作,一般需要这样几个步骤:

- ① 仔细了解并弄清需要解决的问题,建立模型;
- ② 制定方案,确定算法,必要时绘制程序流程图;
- ③ 编写程序;
- ④ 程序调试或系统测试。

本例是一个非常简单的运算,可以直接进行程序编写。编写程序首先要确定使用什么

样的程序设计语言,目前有很多功能强大、使用方便的高级程序设计语言可供用户选用,但为了更好地说明程序在计算机中的工作过程,在这里使用能够直接对系统硬件进行操作的汇编语言来说明。

首先查阅所使用的微处理器的指令表(或指令系统),它是某种微处理器所能执行的全部操作命令汇总。不同系列的微处理器具有不同的指令表。

假定查到模型机的指令表中有3条指令可以用来求解这个问题。表1-1给出了这3条指令的格式、对应的机器码及其说明。

表 1-1 模型机的指令

指令名称	助记符	机器码	指令长度	操 作
数据传送	MOV A, n	10110000 n	2	把立即数 n 送累加器
加法	ADD A, n	00000100 n	2	把累加器的内容与一个常数 n 相加,结果送到累加器
停机	HLT	11110100	1	CPU 暂停运行

表中第1列为指令的名称。编写程序时,写指令的全名是不方便的,因此,人们给每条指令规定了一个缩写词,或称做助记符,如表1.1第2列所示。第3列为机器码,机器码用二进制或十六进制形式表示,这是计算机真正能够识别的指令形式。第4列为指令长度,说明本指令有几个字节。最后一列说明执行一条指令时所完成的具体操作。

现在来编写 $5+8=?$ 的程序。根据指令表提供的指令,用助记符和十进制数表示的加法运算的程序可表达为:

MOV A,5 ;第一个操作数5送到累加器

ADD A,8 ;把累加器的内容5与第2个数8相加,结果(13)送到累加器

HLT ;停机

但是,此程序还有问题。因为模型机并不认识助记符和十进制数,而只认识用二进制数表示的操作码和操作数。因此,必须把以上程序写成二进制数的形式,即用对应的机器码代替每个助记符,用相应的二进制数代替每个十进制数。

MOV A,5 ;对应成二进制码:10110000(指令码) 00000101(操作数)

ADD A,8 ;对应成二进制码:00000100(指令码) 00001000(操作数)

HLT ;对应成二进制码:11110100(指令码)

整个程序共有3条指令,占用5个字节。由于微处理器和存储器均以字节为单位来处理与存放信息,因此,当把这段程序存入存储器时,需要占用5个存储单元。假设把它存放在存储器的最前面5个单元里,则该程序将占用从00H至04H这5个单元的空间,如图1.10

所示。

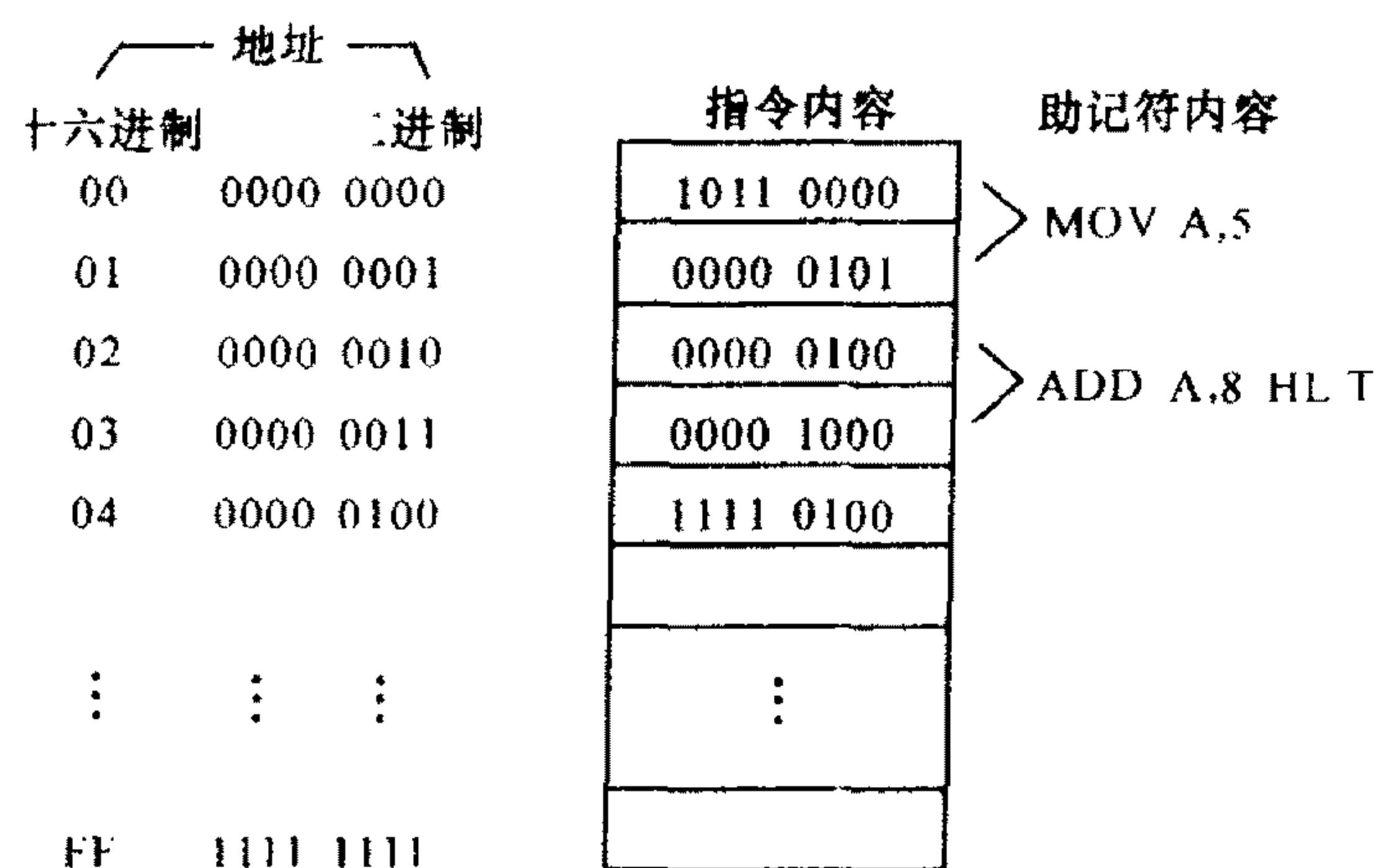


图 1.10 指令在内存中的存放形式

需要强调指出的是：每个内存单元都具有两个和它有关的二进制数，第一个是它的地址，另一个是它的内容。切不可将两种数据的含义相混淆。内存单元的地址是固定的，而内存单元的内容则可以随时由于存入新的内容而改变。

当程序存入存储器并开始执行时，先将第一条指令的首地址 00H 赋给程序计数器 PC，使 PC 指向程序的第一条指令。然后就进入第一条指令的取指阶段，图 1.11 给出了取第一条指令的过程示意图，图中各编号的含义为：

- ① 把 PC 的内容 00H 送到地址寄存器 AR。
- ② 一旦 PC 的内容可靠地送入 AR 后，PC 自动加 1，即由 00H 变为 01H（注意，此时 AR 的内容并没有变化）。
- ③ 把地址寄存器 AR 的内容 00H 放在地址总线上，并送至存储器，经地址译码器译码，选中相应的 00H 单元。
- ④ CPU 的控制器发出读命令。
- ⑤ 在读命令控制下，把所选中的 00H 单元中的内容即第一条指令的操作码 B0H 读到数据总线 DB。
- ⑥ 把读出的内容 B0H 经数据总线送到数据寄存器 DR。
- ⑦ 取指阶段的最后一步是指令译码。因为取出的是指令的操作码，故数据寄存器 DR 把它送到指令寄存器 IR，然后再送到指令译码器 ID。

这就完成了第一条指令的取指阶段。

然后转入第一条指令的执行阶段。经过对操作码 B0H 译码后，CPU“识别”出这个操作码就是“MOV A, n”指令，即把下一个内存单元中的操作数送入累加器 A 的双字节指令，所以，执行第一条指令就必须把指令第二字节中的操作数取出来。取指令第二字节的过程如

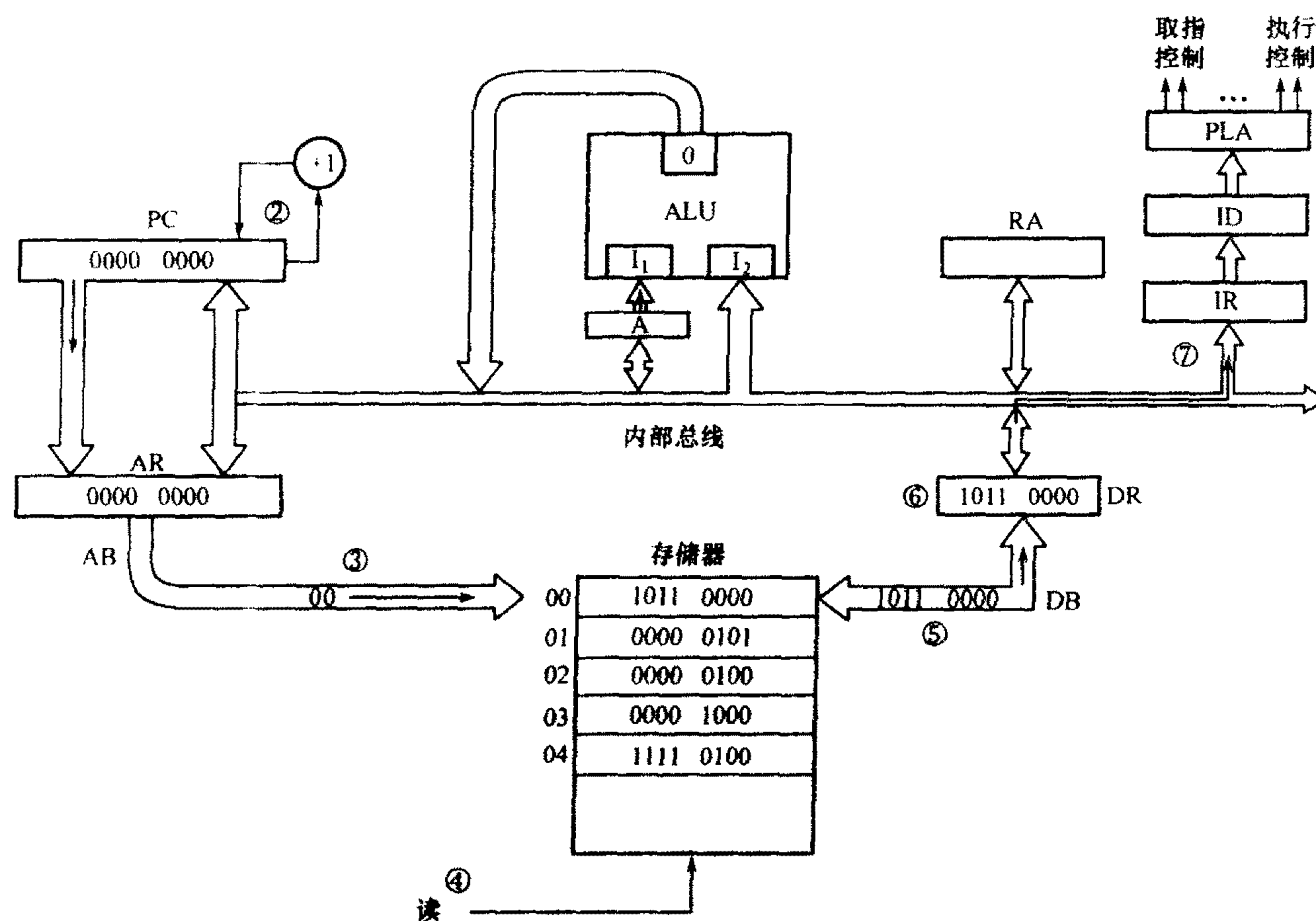


图 1.11 取第一条指令操作码的操作步骤

图 1.12 所示。图中各编号的含义分别为：

- ① 把 PC 的内容 01H 送到地址寄存器 AR。
- ② 当 PC 的内容可靠地送到 AR 后, PC 自动加 1, 变为 02H。但这时 AR 中的内容为 01H, 并未变化。
- ③ 地址寄存器通过地址总线把地址 01H 送到存储器的地址译码器, 经过译码, 选中相应的 01H 单元。
- ④ CPU 的控制器发出读命令。
- ⑤ 在读命令控制下, 将选中的 01H 单元的内容 05H 读到数据总线 DB 上。
- ⑥ 通过 DB 把读出的内容送到数据寄存器 DR。
- ⑦ 因 CPU 根据该条指令具有的字节数已知这时读出的是操作数, 且指令要求把它送到累加器 A, 故由数据寄存器 DR 取出的内容就通过内部数据总线送到累加器 A, 于是第一条指令执行阶段完毕, 数 05H 被取入累加器 A 中, 并进入第二条指令的取指阶段。

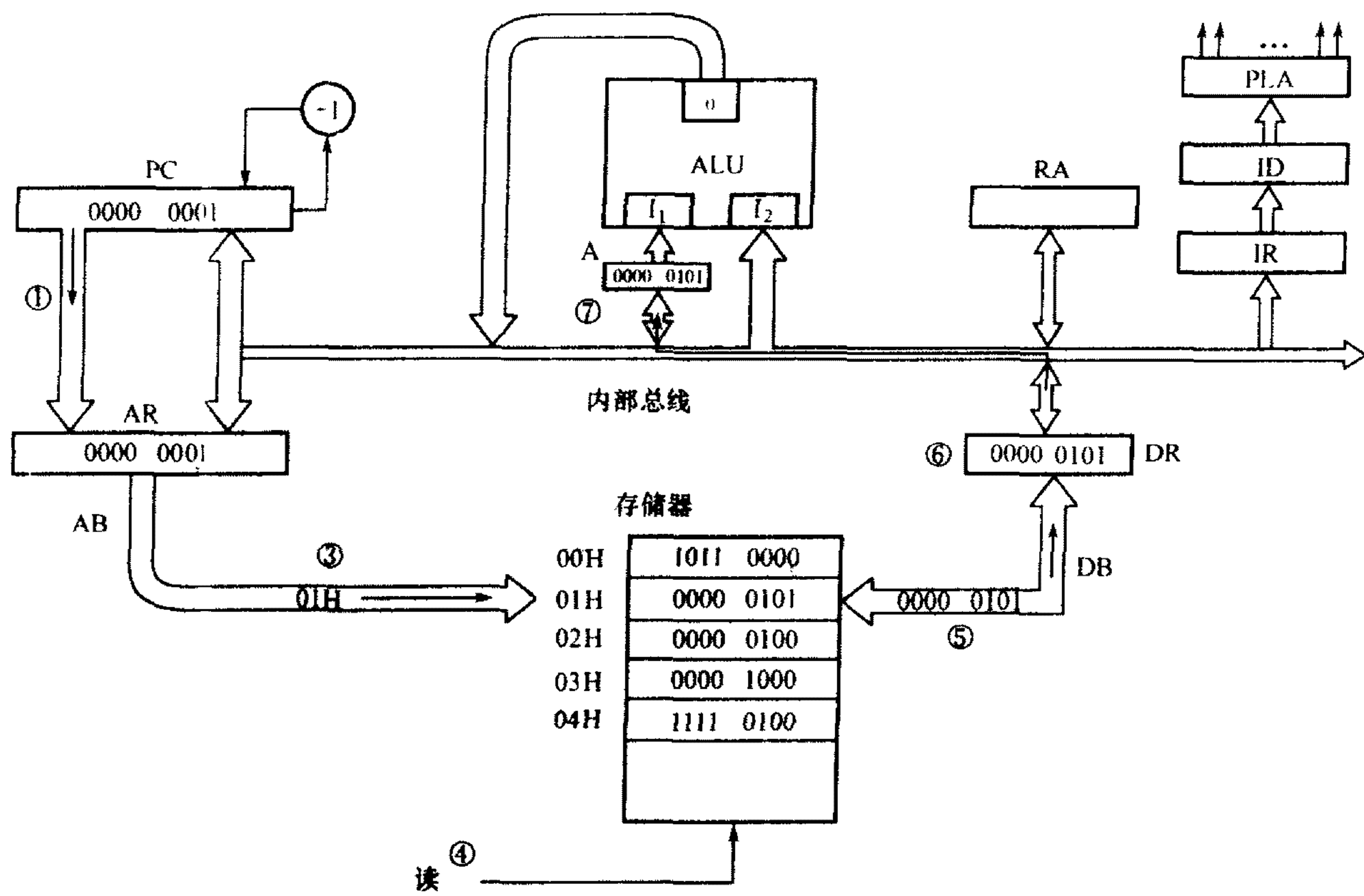


图 1.12 取第一条指令操作数的操作示意图

取第二条指令的过程如图 1.13 所示。它与取第一条指令的过程相同,只是在取指阶段的最后一步读出的指令操作码 04H,由内部数据寄存器 DR 把它送到指令寄存器 IR,经过译码发出相应的控制信息。当指令译码器 ID 对指令译码后,CPU 就“知道”操作码 04H 表示一条加法指令。加法指令把累加器 A 中的内容作为一个操作数,另一个操作数在指令的第二字节中。所以执行第二条指令过程中,必须取出指令的第二字节。

取第二字节操作数及执行指令的过程如图 1.13 和图 1.14 所示。

- ① 把 PC 的内容 03H 送到地址寄存器 AR。
- ② 当把 PC 的内容可靠地送到 AR 后,PC 自动加 1。
- ③ AR 通过地址总线把地址号 03H 送到地址译码器,经过译码,选中相应的 03H 单元。
- ④ CPU 的控制器发出读命令。
- ⑤ 在读命令控制下,把选中的 03H 单元中的内容,即数 08H 读至数据总线上。
- ⑥ 数据被数据总线传送到数据寄存器 DR。

⑦ 因在对指令译码时,CPU 已经知道读出的数据 08H 为操作数,且要将它与已暂存于 A 中的内容 05H 相加,故数据由 DR 通过内部数据总线送至 ALU 的 I_2 输入端。

⑧ A 中的内容送 ALU 的 I_1 输入端,然后执行加法操作。

⑨ 把相加的结果 0DH 由 ALU 的输出端又送到累加器 A 中。

至此,第二条指令的执行阶段结束。A 中存入和数 0DH,而将原有内容 05H 冲掉。接着转入第三条指令的取指阶段。

程序中的最后一条指令是 HLT。可用类似上面的取指过程把它取出。当把 HLT 指令的操作码 F4H 取到数据寄存器 DR 后,因是取指阶段,故 CPU 将操作码 F4H 送指令寄存器 IR,再送指令译码器 ID。经译码,CPU“知道”是暂停指令,于是,控制器停止产生各种控制命令,使计算机停止全部操作。这时,程序已完成 $5+8$ 的运算,并且和数 0DH 已放在累加器 A 中。

以上就是一个 3 条指令的程序在模拟机上的执行过程。从这些叙述和图示中可以体会一台微型计算机的一般工作原理。另外还可以发现,程序中每条指令的长度是不一定相同的,有的占用一个字节,如 HLT 指令;有的则要占用两个字节,如 MOV A, n 与 ADD A, n 指令,所以,计算机中执行的各条指令所需要的时间是不同的。在使用汇编语言设计实时控制系统程序时,应尽量考虑应用字节少的指令。

1.5 计算机常用术语解释

1. 数据单位

1) 位(bit, b)、兆位(Mb)、吉(咖)位(Gb)和太(拉)位(Tb)

“位”是计算机所能处理的最小数据单位,它只能有两种状态:“0”或“1”。一位也叫做一个 bit,可缩写为 b。

Mb 是 Megabit 的缩写,代表 2^{20} 位,即 $1\,024 \times 1\,024$ 位;Gb 是 Gigabit 的缩写,代表 2^{30} 位,即 1 024 Mb;Tb 是 Terabit 的缩写,代表 240 位,即 1 024 Gb。

2) 字节(Byte, B)、千字节(KB)、兆字节(MB)、吉字节(GB)和太字节(TB)

1 个字节包含 8 个二进制位(8 bit)。Byte 通常缩写为 B。要注意的是:大写的 B 表示一个字节,而小写的 b 表示一位,不要写混淆。字节是计算机中存储器容量的基本单位。通常所说的某台计算机的内存容量是 128 M,是表示该机的主存储器有 128 MB,也就是说有 128 M 个存储单元,每个单元包含 8 位二进制数。

在计算机中,1 K 表示 2^{10} ,即 1 024。所以,1 KB = 1 024 B,依此类推,1 MB = 1 024 KB,1 GB = 1 024 MB,1 TB = 1 024 GB。

3) 字(Word)

字在不同的场合有不同的含义。在计算机编程语言中,一个字代表 2 个字节,即 32 位。

对计算机硬件来说,字是 CPU 与输入/输出(I/O)设备和存储器之间传送数据的基本单位。是数据总线的宽度,即数据总线上一次可同时传送的数据的位数。微型计算机的字长有:1 位、4 位、8 位、16 位、32 位和 64 位等。

2. 计算机通信速率单位

1) 波特率(Baud Rate)

波特率是通信信道中信号状态每秒钟变化的次数或每秒传输的信号单元数,是信息传送速率的度量单位。这些状态变化可以是电压大小,也可以是相位角的变化或频率的变化。例如 1 200 波特,表示信号在 1 秒钟内要变换状态 1 200 次。波特率不一定与速率一致,一次状态变化可以传送一个二进制位,也可传送多个二进制位。每一波特可以传送多少位,取决于所使用的标准。

2) 位速率或比特率(Bit Per Second, b/s)

表示每秒传送多少位,它和波特率有关系,但两者并不是一回事。

3) 每秒字符(Character Per Second, CPS)

表示每秒传送多少 ASCII 字符。它的大小与波特率、位速率和校验位、起始位的个数和停止位的个数都有关系。

3. 兼容性

1) 系统兼容

若在一个系统上开发的硬件和软件能够在另外一个系统上成功地运行,则称两个系统是兼容的。

2) 向上兼容

若兼容性是从旧系统到新系统的单项发展,则称为向上兼容。例如,在 PC/XT 机上开发的软件,可以在 PC/AT 机的实模式下正确地运行,反之,则不能正确运行,所以 PC/AT 机与 PC/XT 机是向上兼容的。

习 题 一

- 1.1 微处理器、微型计算机和微型计算机系统有什么不同?
- 1.2 微型计算机系统包括几个部分?简述其硬件系统各部件的主要功能。
- 1.3 微型计算机主机的硬件系统包括哪几个部分?
- 1.4 计算机软件系统包括哪些?
- 1.5 系统软件与应用软件的区别是什么?
- 1.6 冯·诺依曼计算机具有什么特点?

-
- 1.7 计算机系统可分为哪几个层次?
 - 1.8 操作系统的主要功能是什么?
 - 1.9 操作系统的发展经历了几代? 各自具有什么特征?
 - 1.10 DOS 操作系统在结构上有哪些特点?
 - 1.11 简述 DOS 操作系统的装入过程。
 - 1.12 微型计算机的工作过程实际上就是什么过程?
 - 1.13 一条指令的执行主要包括哪两个步骤?

第2章 计算机中的数制和编码

本章介绍二进制、十进制和十六进制方面的基本知识以及各进制之间的转换,并介绍用于 Intel 微处理器的各种数据格式,还介绍 ASCII、BCD、有符号整数和无符号整数及浮点数以及其在机器中的表示方法和基本运算法则。

2.1 计算机中的数制

在日常生活中,人们使用十进制数来进行计数和计算。计算机只能识别由“0”和“1”构成的二进制代码,众所周知,计算机中的数是用二进制表示的;但二进制数既冗长又难以记忆,表示较大的数字时尤为如此。为了阅读和书写方便,或适应某些场合的需要,在计算机中有时也采用十六进制数和十进制数。

2.1.1 常用计数制

1. 十进制数

十进制数的表示方式可概括为“位权表达式”。十进制数中有 0~9 共 10 个数字符号,计数法则为逢十进一。任意十进制数 D ,都可用这 10 个符号的组合来表示。写成权展开式为

$$\begin{aligned}(D)_{10} &= D_{n-1} \times 10^{n-1} + D_{n-2} \times 10^{n-2} + \cdots + D_1 \times 10^1 + D_0 \times 10^0 + D_{-1} \times 10^{-1} \\ &\quad + \cdots + D_{-m} \times 10^{-m} \\ &= \sum_{i=-m}^{n-1} D_i \times 10^i\end{aligned}\quad (2.1)$$

其中, D_i 是 D 的第 i 位的数码,可以是 0~9 10 个符号中的任何一个符号; n 和 m 为正整数, n 表示小数点左边的位数, m 表示小数点右边的位数, 10 为基数,基数是指十进制制允许使用的基本数码的个数; 10^i 称为十进制的位权,简称“权”。十进制数可用下标 10 表示,常可省略。

例 2-1 十进制数 8 756.23 可表示为

$$(8\ 756.23)_{10} = 8 \times 10^3 + 7 \times 10^2 + 5 \times 10^1 + 6 \times 10^0 + 2 \times 10^{-1} + 3 \times 10^{-2}$$

2. 二进制数

二进制数的每一位只有 0 和 1 两个数字符号,且计数法则为逢二进一。一个二进制数 B 按权展开表示为

$$\begin{aligned}(B)_2 &= B_{n-1} \times 2^{n-1} + B_{n-2} \times 2^{n-2} + \cdots + B_0 \times 2^0 + B_{-m} \times 2^{-m} \\ &= \sum_{i=-m}^{n-1} B_i \times 2^i\end{aligned}\quad (2.2)$$

其中, B_i 只能取 1 或 0, 2 为基数, 2^i 为二进制的权, m 、 n 的含义与十进制表达式相同。一个二进制数通常用下标 2 表示,以有别于其他进位计数制。

例 2-2 二进制数 1110.01 可表示为

$$(1110.01)_2 = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}$$

3. 十六进制数

十六进制数共需要 16 个数字符号, 0~9 及 A~F, 其计数法则为逢十六进一。一个十六进制数 H 按权展开为

$$\begin{aligned}(H)_{16} &= H_{n-1} \times 16^{n-1} + H_{n-2} \times 16^{n-2} + \cdots + H_0 \times 16^0 + H_{-1} \times 16^{-1} \\ &\quad + \cdots + H_{-m} \times 16^{-m} \\ &= \sum_{i=-m}^{n-1} H_i \times 16^i\end{aligned}\quad (2.3)$$

其中, H_i 的取值在 0~F 的范围内, 16 为基数, 16^i 为十六进制数的权; m 、 n 的含义与上述相同。十六进制数通常用下标 16 表示。

例 2-3 十六进制数 34A.5H 可表示为

$$(34A.5)_{16} = 3 \times 16^2 + 4 \times 16^1 + A \times 16^0 + 5 \times 16^{-1}$$

由于 $2^4 = 16$, 也就是说 1 位十六进制数恰好可用 4 位二进制数来表示, 所以, 在计算机应用中, 常采用二—十六缩写方式, 虽然机器只能识别二进制数, 但在数字的表达上更广泛地采用十六进制数。

计算机中常用的二进制数、十六进制数和十进制数之间的关系如表 2-1 所示。

表 2-1 数制对照表

十进制数	二进制数	十六进制数	十进制数	二进制数	十六进制数
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B

续表

十进制数	二进制数	十六进制数	十进制数	二进制数	十六进制数
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

4. 其他进制数

除以上介绍的二、十和十六进制三种常用的进位计数制外,计算机中还可能用到八进制数,这里不再详细介绍了。下面给出任一进位制数的权展开式的一般形式。

一般地,对任意一个 K 进制数 S 都可表示为

$$\begin{aligned}
 (S)_K &= S_{n-1} \times K^{n-1} + S_{n-2} \times K^{n-2} + \cdots + S_0 \times K^0 + S_{-1} \times K^{-1} + \cdots + S_{-m} \times K^{-m} \\
 &= \sum_{i=-m}^{n-1} S_i \times K^i
 \end{aligned} \quad (2.4)$$

其中, S_i 是 S 的第 i 位的数码,可以是所选定的 K 个符号中的任何一个; n 和 m 的含义同上; K 为基数, K^i 称为 K 进制数的权。

除了用基数作为下标来表示数的进制外,还可以在数的后面加上字母 B、H、D 来分别表示二进制数、十六进制数和十进制数,如 11000101B、2C0FH、1300D 等。惟有十进制数后面的 D 也可以省略。

2.1.2 各数制之间的转换

人们习惯使用十进制数,计算机内部采用的是二进制数,而编写程序又多采用十六进制数,那么计算机在运算和处理时必然涉及不同进位计数制之间进行转换的问题。

1. 其他进制数到十进制数的转换

非十进制数转换为十进制数,只要将它们按相应的权展开表达式,再按十进制运算规则求和,即可得到对应的十进制数。

例 2-4 将二进制数 1011.011 转换为十进制数。

解 根据二进制数的权展开式,有

$$(1011.011)_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} = (11.375)_{10}$$

例 2-5 将十六进制数 85.CH 转换为十进制数。

解 根据十六进制数的权展开式,有

$$(85.C)_{16} = 8 \times 16^1 + 5 \times 16^0 + C \times 16^{-1} = 8 \times 16^1 + 5 \times 16^0 + 12 \times 16^{-1} = (133.75)_{10}$$

2. 十进制数转换为其他进制数

1) 十进制数转换为二进制数

十进制数整数和小数部分应分别进行转换。整数部分转换为二进制数时采用“除基取余”法,二进制数基数为2,即连续除2,并取余数作为结果,直至商为0,得到的余数从低位到高位依次排列,即得到转换后二进制数的整数部分;对小数部分,则用“乘基取整”法,即对小数部分连续用2乘,以最先得到的乘积的整数部分为最高位,直至达到所要求的精度或小数部分为0为止,转换的结果的整数和小数部分是从小数点开始分别向高位和向低位逐步扩展。

例 2-6 将十进制数 124.25 转换为等值的二进制数。

解

整数部分	小数部分
124/2 = 62.....余数 = 0 (最低位)	0.25 × 2 = 0.5.....整数 = 0 (最高位)
62/2 = 31.....余数 = 0	0.5 × 2 = 1.0.....整数 = 1
31/2 = 15.....余数 = 1	
15/2 = 7.....余数 = 1	
7/2 = 3.....余数 = 1	
3/2 = 1.....余数 = 1	
1/2 = 0.....余数 = 1	

从而得到转换结果

$$(124.25)_{10} = (1111100.01)_2$$

2) 十进制数转换为十六进制数

与十进制数转换为二进制数的方法类似,整数部分按“除 16 取余”的方法进行,而小数部分按“乘 16 取整”的方法进行。

例 2-7 将十进制数 455.6875 转换为十六进制数。

解

整数部分	小数部分
455/16 = 28.....余数 = 7	0.6875 × 16 = 11.0000.....整数 = (11) ₁₀ = (B) ₁₆
28/16 = 1.....余数 = C	
1/16 = 0.....余数 = 1	

也可将十进制数先转换为二进制数,再转换为十六进制数。

3. 二进制数与十六进制数之间的转换

由于 $2^4 = 16$,十六进制数与二进制数间的相互转换非常容易。将二进制数转换为十六

进制数的方法是：从小数点开始分别向左和向右把整数和小数部分每 4 位分为一组。若整数最高位的一组不足 4 位，则在其左边补零；若小数最低位的一组不足 4 位，则在其右边补零。然后将每组二进制数用对应的十六进制数代替，则得到转换结果。

例 2-8 将二进制数 110100110.101101B 转换为十六进制数。

解

二进制数	<u>0001</u>	<u>1010</u>	<u>0110.</u>	<u>1011</u>	<u>0100</u>
	↓	↓	↓	↓	↓
十六进制数	1	A	6.	B	4

所以有

$$(110100110.101101)_2 = (1A6.B4)_{16}$$

十六进制数转换为二进制数的方法与上述过程相反，即用 4 位二进制代码取代对应的 1 位十六进制数。

例 2-9 将十六进制数 7AD4.6DH 转换为二进制数。

解

十六进制数	7	A	D	4.	6	D
	↓	↓	↓	↓	↓	↓
二进制数	0111	1010	1101	0100.	0110	1101

所以有

$$(7AD4.6D)_{16} = (0111101011010100.01101101)_2$$

2.2 无符号二进制数的运算

2.2.1 二进制的算术运算

1. 加法运算

二进制数只有 0 和 1 两个数，其运算规则比十进制数简单得多，二进制的加法运算“逢二进一”，遵循如下法则：

$$0 + 0 = 0, \quad 0 + 1 = 1, \quad 1 + 0 = 1, \quad 1 + 1 = 0(\text{有进位})$$

例 2-10 计算 $10100110B + 01011101B = (?)B$ 。

解

进位		1	1	1	1	1	1	0	0	0
被加数		1	0	1	0	0	1	1	0	
加数	+	0	1	0	1	1	1	0	1	
		<hr/>								
		1	0	0	0	0	0	0	1	1

可得

$$10100110B + 01011101B = 00000011B$$

2. 减法运算

二进制数的减法遵循如下法则：

$$0 - 0 = 0, \quad 1 - 0 = 1, \quad 1 - 1 = 0, \quad 0 - 1 = 1 \text{ (有借位)}$$

例 2-11 计算 $11000100B - 00100101B = (?)B$

解

借位		0	1	1	1	1	1	0	
被减数		1	1	0	0	0	1	0	0
减数	-	0	0	1	0	0	1	0	1
		<hr/>							
		1	0	0	1	1	1	1	1

可得

$$11000100B - 00100101B = 10011111B$$

3. 乘法运算

二进制数的乘法遵循如下法则：

$$0 \times 0 = 0, \quad 0 \times 1 = 0, \quad 1 \times 0 = 0, \quad 1 \times 1 = 1$$

即仅当两个 1 相乘时结果为 1, 否则结果为 0。所以, 二进制数的乘法也是非常简单的。若乘数位为 1, 就将被乘数加于中间结果; 若乘数位为 0, 则加 0 于中间结果, 只是在相加时, 要将每次中间结果的最后一位与相应的乘数位对齐。

例 2-12 求两个二进制数 $1100B$ 与 $1010B$ 的乘积。

解法一 按照十进制的乘法过程有

	1	1	0	0	被乘数			
×	1	0	1	0	乘数			
<hr/>					部分积			
	0	0	0	0				
	1	1	0	0				
	0	0	0	0				
	1	1	0	0				
<hr/>								
	1	1	1	1	0	0	0	乘积

可得

$$1100\text{B} \times 1010\text{B} = 1111000\text{B}$$

解法二 采用移位加的方法,则有

乘数	被乘数	部分积
1 0 1 0	1100	0000
乘数为 0, 不加被乘数, 被乘数左移 1 位	11000	0000
乘数为 1, 加被乘数到部分积上, 并将 被乘数左移 1 位	110000	11000
乘数为 0, 不加被乘数, 被乘数左移 1 位	1100000	11000
乘数为 1, 加左移后的被乘数到部分积上		+ 1100000
		<u>1111000</u>

即可得

$$1100\text{B} \times 1010\text{B} = 1111000\text{B}$$

由方法二可看出,二进制的乘法运算可以转换为加法和移位的运算。事实上计算机中乘法运算就是这样做的。每左移 1 位,相当于乘以 2,而左移 n 位就相当于乘以 2^n 。

4. 除法运算

除法是乘法的逆运算,所以二进制数的除法运算也可转换为减法和右移运算。每右移 1 位相当于除以 2,右移 n 位就相当于除以 2^n 。

2.2.2 无符号数的表示范围

1. 无符号二进制数的表示范围

一个 n 位的无符号二进制数 X ,它可表示的数的范围为 $0 \leq X \leq 2^n - 1$ 。

对于一个 8 位的二进制数,即 $n = 8$,其表示范围为 $0 \sim 2^8 - 1$,即 $00\text{H} \sim \text{FFH}$ ($0 \sim 255$)。若运算结果超出数的可表示范围,则会产生溢出,结果将不正确。

例 2-13 计算 $11011011\text{B} + 01001101\text{B} = (?)\text{B}$

解

$$\begin{array}{r}
 11011011 \\
 + 01001101 \\
 \hline
 100101000
 \end{array}$$

由上式可得,上面两个 8 位二进制数相加的结果为 9 位,超出了 8 位数的表示范围。若仅取 8 位字长 (00101000B),结果显然不正确,这种情况就称为溢出。事实上, $(11011011)_2 =$

$(219)_{10}, (01001101)_2 = (77)_{10}$, 则 $219 + 77 = 296$, 大于 8 位二进制数所能表示的最大值 255, 所以最高位的进位(代表了 256)给丢失了, 这样最后的结果就是 $296 - 256 = 40$, 即 $00101000B$ 。

2. 无符号二进制数的溢出判断

设无符号二进制数加法(或减法)中最高有效位 D_i 的进(借)位为 C_i , 则两个无符号二进制数相加(或相减)时, 若最高有效位 D_i 产生进位(或相减有借位), 即 $C_i = 1$, 则产生溢出。例如在例 2-13 中, 两个 8 位无符号二进制数相加, 最高有效位(即 D_7 位)产生了进位 C_7 , 结果就出现溢出。

2.2.3 二进制数的逻辑运算

逻辑运算与算术运算不同, 不是将一个二进制数的所有位都作为一个数值整体来考虑, 而是对二进制数按位进行操作, 也就是说逻辑运算没有进位和借位。基本逻辑运算包括与、或、非及异或四种运算。

1. 与运算

与运算的规则是按位相与。与运算符一般用符号“ \wedge ”表示。其规则如下:

$$1 \wedge 1 = 1, \quad 1 \wedge 0 = 0, \quad 0 \wedge 1 = 0, \quad 0 \wedge 0 = 0$$

即参加与操作的两位中只要有一位为 0, 则与的结果就为 0, 仅当两位均为 1 时, 其结果才为 1。相当于按位相乘(但不进位), 又叫做“逻辑乘”。

例 2-14 计算 $11011010B \wedge 10010110B = (?)B$ 。

解

$$\begin{array}{r} 11011010 \\ \wedge 10010110 \\ \hline 10010010 \end{array}$$

即

$$11011010B \wedge 10010110B = 10010010B$$

2. 或运算

或运算又叫做“逻辑加”, 其规则为按位相或。或运算符一般用符号“ \vee ”表示。其规则如下:

$$0 \vee 0 = 0, \quad 0 \vee 1 = 1, \quad 1 \vee 0 = 1, \quad 1 \vee 1 = 1$$

即参加或操作的两位中仅当两位均为 0 时, 其结果才为 0, 只要有一位为 1, 则或的结果就为 1。(比较二进制数的或运算规则和加法运算规则有什么不同?)

例 2-15 计算 $11011001B \vee 10010110B = (?)B$ 。

解

$$\begin{array}{r}
 1\ 1\ 0\ 1\ 1\ 0\ 0\ 1 \\
 \vee\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0 \\
 \hline
 1\ 1\ 0\ 1\ 1\ 1\ 1\ 1
 \end{array}$$

即

$$11011001B \vee 10010110B = 11011111B$$

3. 非运算

非运算的规则为按位取反,即1的非为0,而0的非为1。非属于单边运算,即只有一个运算对象,其运算符为一上横线(—)。

$$\bar{1} = 0 \quad \bar{0} = 1$$

例2-16 求11011001B的非运算。

解 只要对11011001B按位求反,即可得

$$\overline{11011001B} = 00100110B$$

4. 异或运算

异或运算相当“按位相加”(不进位),进行异或操作的两个二进制位不同时,结果就为1;两位相同时,结果为0。异或运算符用符号 \oplus 表示。其规则如下:

$$0 \oplus 0 = 0, \quad 1 \oplus 1 = 0, \quad 0 \oplus 1 = 1, \quad 1 \oplus 0 = 1$$

例2-17 计算 $11010011B \oplus 10100110B = (?)B$ 。

解:

$$\begin{array}{r}
 1\ 1\ 0\ 1\ 0\ 0\ 1\ 1 \\
 \oplus\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 0 \\
 \hline
 0\ 1\ 1\ 1\ 0\ 1\ 0\ 1
 \end{array}$$

即

$$11010011B \oplus 10100110B = 01110101B$$

比较有趣的是:二进制数的异或运算可以看做不进位的“按位加”,也可以看做不借位的“按位减”。

2.2.4 基本逻辑门及常用逻辑部件

本小节介绍几种常用的计算机基本逻辑部件,已经学过数字电路的读者可跳过本节。对这些逻辑部件,仅是从应用的角度出发,只介绍它们的逻辑功能和外部引线连接,而不涉及其内部的电路构成。

1. 与门(AND Gate)

与门是对多个逻辑变量(取值只有“0”和“1”)进行与运算的门电路。对两个逻辑变量 A

和 B 的“与”操作,其结果 Y 可表示为

$$Y = A \wedge B$$

它们的关系也可从表 2-2 所示的真值表中清楚地了解到。即仅在输入 A 和 B 均为 1 时,输出 Y 才为 1;A 和 B 中只要一个为 0,则 Y 就等于 0。从电路的角度来说,若采用正逻辑,则仅当与门的输入 A 和 B 都是高电平时,输出 Y 才是高电平,否则 Y 就输出低电平。

在电路连接上,与门常用图 2.1 所示的逻辑符号表示。

表 2-2 与门的真值表

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

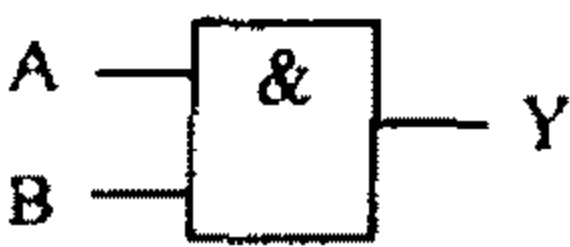


图 2.1 与门的逻辑符号

2. 或门(OR Gate)

或门是对多个逻辑变量进行或运算的门电路。对两个逻辑变量 A 和 B 的或操作,它们的逻辑关系可用如下的运算表达式表示:

$$Y = A \vee B$$

即两个输入变量 A 和 B 中任意一个为 1,输出 Y 就为 1;仅当 A 和 B 都为 0 时,Y 才为 0。从电路的角度来说,当或门的输入 A 和 B 只要有一个是高电平,输出 Y 就为高电平,否则 Y 就输出低电平。

或门的逻辑符号如图 2.2 所示,其真值表见表 2-3。

表 2-3 或门的真值表

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



图 2.2 或门的逻辑符号

3. 非门(NOT Gate)

非门又称为反相器,是对单一逻辑变量进行非运算的门电路,其输入变量 A 与输出变量 Y 之间的关系可用下式表示:

$$Y = \bar{A}$$

非运算也称求反运算,变量 A 上的上划线“—”在数字电路中表示反相之意。逻辑符号图中的小圆圈表示非(本书将始终采用这种表示方法),非门的逻辑符号如图 2.3 所示,其真值表见表 2-4。

表 2-4 非门的真值表

A	Y
0	1
1	0

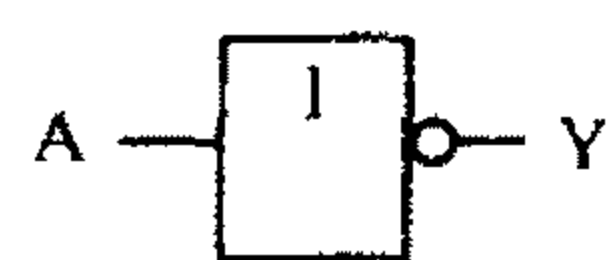


图 2.3 非门的逻辑符号

4. 与非门(NAND Gate)

与门与非门结合形成与非门。若输入变量为 A 和 B,则先对 A 和 B 进行与运算,再对结果进行非运算,运算表达式为

$$Y = \overline{A \wedge B}$$

与非门的逻辑符号如图 2.4 所示,其真值表见表 2-5。

表 2-5 与非门的真值表

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

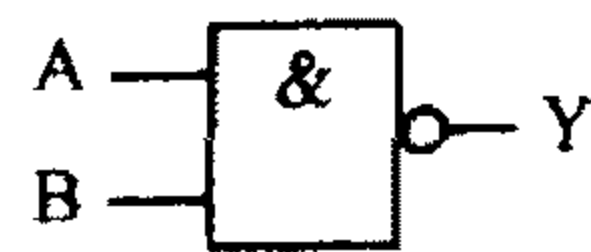


图 2.4 与非门的逻辑符号

5. 或非门(NOR Gate)

和与非门类似,或非门是或门与非门的结合。即先对输入 A 和 B 进行或运算,再对其结果进行非运算,其运算表达式为

$$Y = \overline{A \vee B}$$

或非门的逻辑符号如图 2.5 所示,真值表见表 2-6。

表 2-6 或非门的真值表

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

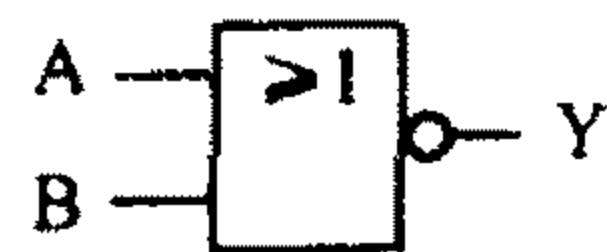


图 2.5 或非门的逻辑符号

6. 译码器

在计算机硬件系统中,常常需要通过一定的控制电路从一系列地址信号中选出对某一芯片的片选信号,这个控制电路称为译码电路,它所对应的逻辑部件就称为译码器。

译码器的种类很多,这里仅介绍一种常用的 3 线 - 8 线译码器 74LS138。74LS138 的引脚如图 2.6 所示。图中 G_1 、 $\overline{G_{2A}}$ 、 $\overline{G_{2B}}$ 为译码器的三个使能输入端,它们共同决定了译码器当前是否被允许工作。当 $G_1 = 1$, $\overline{G_{2A}} = \overline{G_{2B}} = 0$ 时,译码器处于使能状态(Enable),否则就被禁止(Disable)。C、B、A 为译码器的 3 条输入线(输入的 3 位二进制代码分别代表了 8 种不同的状态),它们的不同的状态组合,决定了 8 个输出端 $Y_0 \sim Y_7$ 的状态。74LS138 的功能见表 2-7,表中电平为正逻辑,即高电平表示逻辑“1”,低电平表示逻辑“0”,“x”表示不定。

表 2-7 74LS138 功能表

使能端			输入端			输出端							
G_1	$\overline{G_{2A}}$	$\overline{G_{2B}}$	C	B	A	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
x	1	1	x	x	x	1	1	1	1	1	1	1	1
0	x	x	x	x	x	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	0	1	1	0	1	1	1	1	1	1
1	0	0	0	1	0	1	1	0	1	1	1	1	1
1	0	0	0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	0	0	1	1	1	1	0	1	1	1
1	0	0	1	0	1	1	1	1	1	1	0	1	1
1	0	0	1	1	0	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0

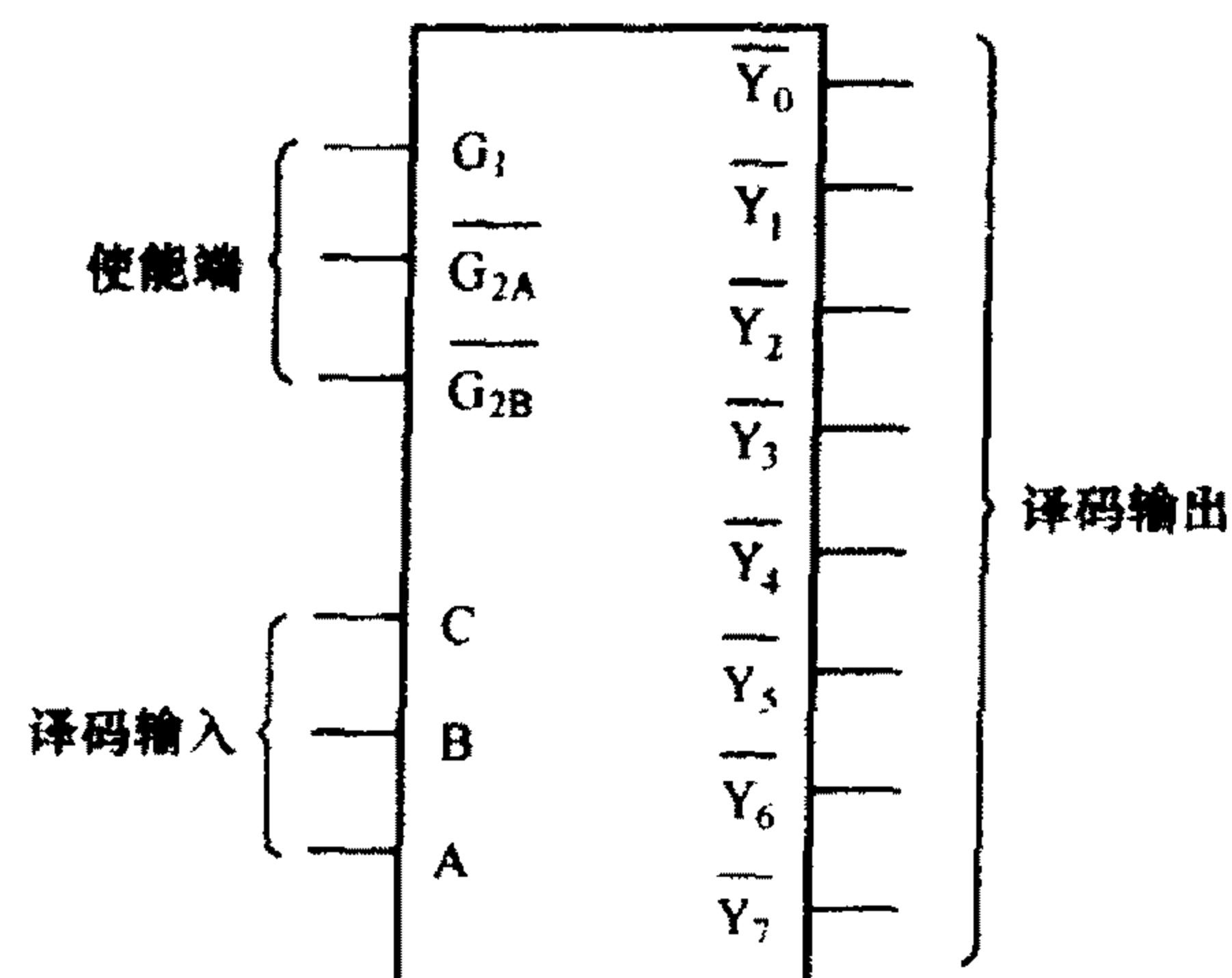


图 2.6 74LS138 的引脚功能图

2.3 带符号二进制数的表示及运算

实际生活中数是有正有负的,通常用符号“+”和“-”表示正数和负数,前面讨论的是未涉及符号的二进制数,称为无符号数。但在计算机中,所使用的数均是由0和1两个数字组成,而数的正、负号也只能由0和1来表示。也就是说符号需要数字化,通常规定,一个有符号数的最高位代表符号,该位为“0”表示正,该位为“1”表示负。以8位字长为例, D_7 为符号位, $D_6 \sim D_0$ 为数值位,若字长为16位,则 D_{15} 为符号位, $D_{14} \sim D_0$ 为数值位。

例 2-18 $+0010101B$ 在计算机中可表示为 $00010101B$,即十进制数的 $+21$; $-0010101B$ 在计算机中可表示为 $10010101B$,即十进制数的 -21 。

符号被数值化了的数称为机器数,如 00010101 和 10010101 就是机器数,而把原来的数值称为机器数的真值。如 $+0010101$ 和 -0010101 。下面介绍带符号机器数的表示方法及它们的运算规则。

2.3.1 带符号数的表示方法

在计算机中一个带符号数有三种表示方法,即原码、反码和补码。它们均是由符号位和数值部分组成,而且符号位有相同的表示方法,即用1来表示负号(-),用0来表示正号(+).下面仅说明原码、反码和补码的数值部分的表示方法。

1. 原码

真值 X 的原码记为 $[X]_{\text{原}}$ 。在原码表示法中,不论数的正负,数值部分均保持原真值不变。

例 2-19 已知真值 $X = +24$, $Y = -24$,求 $[X]_{\text{原}}$ 和 $[Y]_{\text{原}}$ 。

解 因为 $(+24)_{10} = +0011000B$, $(-24)_{10} = -0011000B$,根据原码表示法,有

$$\begin{array}{ccc} [X]_{\text{原}} = 0 & 0011000 & [X]_{\text{原}} = 1 \quad 0011000 \\ \text{符号位} & \text{数值部分} & \text{符号位} \quad \text{数值部分} \end{array}$$

注意,在原码表示法中,真值0的原码可表示为两种不同的形式,即 $+0$ 和 -0 。

$$[+0]_{\text{原}} = 00000000$$

$$[-0]_{\text{原}} = 10000000$$

原码表示法的优点是简单,易于理解,与真值间的转换较为方便;它的缺点是进行加、减运算时较麻烦,不仅要考虑是做加法还是做减法,而且要考虑数的符号和绝对值的大小,这不仅使运算器的设计较为复杂,而且降低了运算器的运算速度。

当机器数的最高位为 1 时,这个数字 1 表示的是负号,因此,二进制数 $X = X_{n-1}X_{n-2}\cdots X_1X_0$ (n 代表二进制数的字长),其原码表示的严格定义是

$$[X]_{\text{原}} = \begin{cases} X, & 2^{n-1} > X \geq 0 \\ 2^{n-1} - X = 2^{n-1} + |X|, & 0 \geq X > -2^{n-1} \end{cases} \quad (2.5)$$

2. 反码

真值 X 的反码记为 $[X]_{\text{反}}$ 。对正数而言,其表示方法同原码,即数值部分与真值相同;但对负数而言,其反码的数值部分为真值的各位按位取反。

例 2-20 已知真值 $X = +24$, $Y = -24$, 求 $[X]_{\text{反}}$ 和 $[Y]_{\text{反}}$ 。

解 因为 $(+24)_{10} = +0011000\text{B}$, $(-24)_{10} = -0011000\text{B}$, 根据反码表示法,有

$$[X]_{\text{反}} = 00011000, \quad [Y]_{\text{反}} = 11100111$$

在反码表示法中,同原码一样,数 0 也有两种表示形式:

$$[+0]_{\text{反}} = 00000000, \quad [-0]_{\text{反}} = 11111111$$

由上可见,用原码和反码表示带符号数时,数值 0 的表示都不是惟一的,这样使用起来很不方便。目前在微处理器中已不使用这两种表示方法。

若二进制数 $X = X_{n-1}X_{n-2}\cdots X_1X_0$, 则反码表示的严格定义是

$$[X]_{\text{反}} = \begin{cases} X, & 2^{n-1} > X \geq 0 \\ 2^{n-1} + X, & 0 \geq X > -2^{n-1} \end{cases} \quad (2.6)$$

3. 补码

补码是根据同余的概念得出的。由同余的概念可知,对一个数 X ,有

$$X + nK = X \pmod{K} \quad (2.7)$$

式中 K 为模数, n 为任意整数。即在模的意义下,数 X 就等于其本身加上它的模的任意整数倍之和。若设 n 为 1, $K = 2^n$, 则有

$$X = X + 2^n \pmod{2^n}$$

即

$$X = \begin{cases} X, & 2^{n-1} > X \geq 0 \\ 2^n + X = 2^n - |X|, & 0 > X \geq -2^{n-1} \end{cases} \pmod{2^n} \quad (2.8)$$

实际上,上式就是补码表示的定义。

真值 X 的补码记为 $[X]_{\text{补}}$ 。对正数来说,其表示方法同原码,即数值部分与真值相同(即当 $X \geq 0$ 时, $[X]_{\text{补}} = [X]_{\text{反}} = [X]_{\text{原}}$);但对负数而言,其补码表示的数值部分为真值的各位按位取反再加 1(即当 $X < 0$ 时, $[X]_{\text{补}} = [X]_{\text{反}} + 1$)。

例 2-21 已知真值 $X = +0110100$, $Y = -0110100$, 求 $[X]_{\text{补}}$ 和 $[Y]_{\text{补}}$ 。

解 这里 $X > 0$, 所以有

$$[X]_{\text{补}} = 00110100$$

而 $Y < 0$, 所以有

$$[Y]_{\text{补}} = [Y]_{\text{反}} + 1 = 11001011 + 1 = 11001100$$

与原码和反码不同, 数 0 的补码表示是惟一的。由补码的定义可知

$$[+0]_{\text{补}} = [+0]_{\text{反}} = [+0]_{\text{原}} = 00000000$$

$$[-0]_{\text{补}} = [-0]_{\text{反}} + 1 = 11111111 + 1 = 00000000 (\text{mod } 2^8)$$

即对 8 位字长来讲, 最高位的进位 (2^8) 按模 256 运算被舍掉, 所以

$$[+0]_{\text{补}} = [-0]_{\text{补}} = 00000000$$

另外要注意的是, 对 8 位二进制数 10000000 (16 位二进制数为 1000000000000000, 依此类推), 在补码中定义为 -128 (16 位二进制数 1000000000000000 定义为 -32768), 而在原码中表示 -0 , 在反码中表示 -127 。

2.3.2 补码数与十进制数之间的转换

要把一个用补码表示带符号的二进制数转换为十进制数, 首先应求出它的真值, 然后再进行二—十进制转换即可。下面就来看一下, 若已知一个数的补码, 如何求出它的真值的例子。

由前面的讨论可知, 对于一个用补码表示的 8 位二进制数, 当其符号位为“0”时, 表示是一个正数, 这时它的补码就等于它的原码, 即真值就是它的数值部分, 也就是说, 除符号位之外的其余 7 位就是此数的二进制数值。

例 2-22 已知 $[X]_{\text{补}} = 00111110$, 求 X 的真值。

解 因为补码 00101110 的符号位为 0, 亦即它是一个正数, 它的数值部分就是它的真值。即

$$X = +0111110 = (+62)_{10}$$

而对于一个用补码表示的负数 (符号位为“1”), 求其真值的方法是将此补码数再求一次补, 即将除符号位外的低 7 位按位取反, 再在最低位加 1, 所得结果才是它的真值。或者可以这样理解, 将这个数连同符号位一起按位求反加 1, 得到的即是该数的绝对值, 在前面加上负号即为真值。

例 2-23 已知 $[X]_{\text{补}} = 11010010$, 求 X 的真值。

解 因为补码 11010010 的符号位为“1”, 可知它是一个负数。因此, 对其数值部分再求一次补码, 即可得到 X 的真值为

$$X = [[X]_{\text{补}}]_{\text{补}} = [11010010]_{\text{补}} = -0101110 = (-46)_{10}$$

为什么要引进补码的概念呢?这是因为在计算机中,可以将二进制的乘法运算转换为加法和左移运算,相应的除法则可转换为减法和右移运算,加、减、乘、除运算最终可归结为加、减和移位三种操作来完成。为节省设备,在计算机中一般只设置加法器而无减法器,这就需要将减法运算转化为加法运算,从而使计算机中的二进制四则运算最终变成加法和移位两种操作。引进补码运算就是用来解决将减法运算转化为加法运算的。

2.3.3 补码的运算

两个 n 位二进制数补码的运算具有如下规则:

(1) 和的补码等于补码之和,即

$$[X + Y]_{\text{补}} = [X]_{\text{补}} + [Y]_{\text{补}}$$

(2) 差的补码等于补码之差,即

$$[X - Y]_{\text{补}} = [X]_{\text{补}} - [Y]_{\text{补}}$$

(3) 差的补码也等于第一个数的补码与第二个数的负数的补码之和,即

$$[X - Y]_{\text{补}} = [X]_{\text{补}} + [-Y]_{\text{补}}$$

这里, $[-Y]_{\text{补}}$ 称为对补码数 $[Y]_{\text{补}}$ 变补,变补的规则为:对 $[Y]_{\text{补}}$ 的每一位(包括符号位)按位取反加 1,则结果就是 $[-Y]_{\text{补}}$ 。当然也可以直接对 $-Y$ 求补码,结果也是一样的。

例 2-24 设 $X = +66$, $Y = +51$, 求 $[X - Y]_{\text{补}} = ?$

解 可利用补码运算规则 3 求解 $[X - Y]_{\text{补}} = [X]_{\text{补}} + [-Y]_{\text{补}}$ 。

先求 $[X]_{\text{补}}$ 和 $[-Y]_{\text{补}}$:

$$X = (+66)_{10} = (+1000010)_2, \quad [X]_{\text{补}} = 01000010$$

$$-Y = (-51)_{10} = (-0110011)_2, \quad [-Y]_{\text{补}} = 11001101$$

再求 $[X]_{\text{补}} + [-Y]_{\text{补}}$:

$$\begin{array}{r} 01000010 \\ + 11001101 \\ \hline 10000111 \\ \uparrow \\ \text{自然丢失} \end{array}$$

所以

$$[X - Y]_{\text{补}} = 00001111 = (+15)_{10}$$

在字长为 8 位的机器中,从第 8 位向上的进位是自然丢失的,故本例中做减法运算的结果与用补码做加法运算的结果是相同的,都是十进制数 15。由此说明,当两个带符号数用补码表示时,减法运算可转换为加法运算。

例 2-25 设 $X = +48, Y = +63$, 求 $[X - Y]_{\text{补}} = ?$

$$[X - Y]_{\text{补}} = [X]_{\text{补}} + [-Y]_{\text{补}}$$

$$X = (+48)_{10} = (+0110000)_2, [X]_{\text{补}} = 00110000$$

$$-Y = (-63)_{10} = (-0111111)_2, [-Y]_{\text{补}} = 11000001$$

$$\begin{array}{r} 00110000 \\ + 11000001 \\ \hline 11110001 \end{array}$$

所以

$$[X - Y]_{\text{补}} = 11110001$$

由补码运算规则知,两补码相加的结果即为和的补码,现在和的符号位为 1,和一定为负数。于是将数值部分的后 7 位按位取反后再加 1,得出真值为 -0001111 。故通过补码相加后,和为十进制数 -15 。

还可通过钟表来说明“补”码的概念。假如有一只钟表的时针指在 9 点,若要拨到 5 点,有两种拨法:

① 逆时针拨,倒拨 4 小时, $9 - 4 = 5$ 。

② 顺时针拨,正拨 8 小时, $9 + 8 = 12 + 5 = 17 \bmod 12 = 5$ 。

此处的 12 就是时钟系统中的模(计数系统最大的数),它是自然丢失的,故顺时针拨 8 个字相当于倒拨 4 小时,结果都是 5。

对模 12 而言, $9 - 4 = 9 + 8$,这时就称 8 为 -4 的以 12 为模的补数,即

$$[-4]_{\text{补}} = 12 - 4 = 8$$

这与上面的表达式是一致的。这样就有

$$9 - 4 = 9 + (-4) = 9 + (12 - 4) = 9 + 8 = \underline{12} + 5 = 5$$

↓

模自然丢失

在二进制数系统中,模为 2^n (n 为字长)。若字长为 8 位,其模为 $2^8 = (256)_{10}$ 。

当一个负数用补码表示时,就可以将减法转换为加法来进行计算。如前面的例 2-24 中, $(66 - 51)$ 可写成

$$66 - 51 = 66 + (-51) = 66 + (256 - 51) = 66 + 205 = 256 + 15 = 15 \pmod{256}$$

可见在模为 2^8 的情况下, $(66 - 51)$ 与 $(66 + 205)$ 的结果是相同的。也就是说,对模为 256 来说, -51 与 205 互为补数,这里 -51 的补码二进制数为 11001101,即是十进制数的 205(把 11001101 看成无符号数时为 205,若看成有符号数为 -51 ,正是利用了负数的补码概念,把减法运算转换为加法运算。但要注意,这里负数 $(-X)$ 的补码是利用 $2^8 - X$ 来得到的,实际

上,根据负数的补码的定义 $[X]_{\text{补}} = [X]_{\text{反}} + 1$,即可避免求补过程中的减法运算,使2的补码运算具有了实用价值。

注意:在微型计算机中,凡是涉及带符号数都一定是用补码表示的,所以运算的结果也是用补码表示的。

2.3.4 带符号数运算时的溢出问题

1. 带符号数的表示范围

1) 对8位二进制数,原码、反码和补码所能表示的范围为:

① 原码: $11111111\text{B} \sim 01111111\text{B} (-127 \sim +127)$;

② 反码: $10000000\text{B} \sim 01111111\text{B} (-127 \sim +127)$;

③ 补码: $10000000\text{B} \sim 01111111\text{B} (-128 \sim +127)$ 。

当8位二进制数的运算结果超出以上范围时,就会产生溢出。

(2) 对16位二进制数,原码、反码和补码所能表示的范围为:

① 原码: $\text{FFFFH} \sim 7\text{FFFH} (-32\,767 \sim +32\,767)$;

② 反码: $8000\text{H} \sim 7\text{FFFH} (-32\,767 \sim +32\,767)$;

③ 补码: $8000\text{H} \sim 7\text{FFFH} (-32\,768 \sim +32\,767)$ 。

当16位二进制数的运算结果超出以上范围时,就会产生溢出。

2. 带符号数的运算时的溢出判断

在进行两个有符号数的加减运算时,如果运算结果超出上述的有效范围,就会发生溢出,使计算结果出错。显然,溢出只能出现在两个同符号数相加或两个异符号数相减的情况下。判断一个有符号数的运算是否溢出,在两个同符号数相加或异符号数相减时,有下述规则:

1) 如果次高位向最高位有进位(或借位),而最高位向前无进位(或借位),则结果发生溢出;

2) 反过来,如果次高位向最高位无进位(或借位),而最高位向前有进位(或借位),则结果也发生溢出。

对于8位二进制数,若 D_6 位产生的进位(或借位)记为 C_6 , D_7 位产生的进位(或借位)记为 C_7 ,那么上述两种情况也可表述为

在两个带符号二进制数相加或相减时,若 $C_7 \oplus C_6 = 1$,则结果产生溢出。

例 2-26 用二进制补码计算 $(+76) + (+94) = (?)$ 。

解

$$(+76)_{10} = (+1001100)_2, \quad (+1001100)_{\text{补}} = 01001100$$

$$(+94)_{10} = (+1011110)_2, \quad (+1011110)_{\text{补}} = 01011110$$

0 1 0 0 1 1 0 0 B	76
+ 0 1 0 1 1 1 1 0 B	+ 94
1 0 1 0 1 0 1 0 B	- 86

本例中两个正数相加,其结果(补码)却变成了负值。实际上,是由于 $(+76) + (+94) = +170 > +127$,超出了8位二进制补码的表示范围,从而使结果产生溢出而出错。在计算中,从 $C_6 = 1, C_7 = 0$ 可判断出结果溢出。

例 2-27 用二进制补码计算 $(-78) + (-86) = (?)$ 。

解

$$(-78)_{10} = (-1001110)_2, \quad (-1001110)_{\text{补}} = 10110010$$

$$(-86)_{10} = (-1010110)_2, \quad (-1010110)_{\text{补}} = 10101010$$

1 0 1 1 0 0 1 0 B	- 78
+ 1 0 1 0 1 0 1 0 B	- 86
进位自然丢失 → 1 0 1 0 1 1 1 0 0 B	+ 92

本例中两个负数相加,其结果却变成了正值。事实上, $(-78) + (-86) = -164 < -128$,超出了8位二进制补码的表示范围,从而使结果产生溢出而出错。溢出可从 $C_6 = 0, C_7 = 1$ 判断得知。

以上是两个同符号数相加,当结果超出二进制补码的表示范围时将产生溢出;而对两个异符号数相减同样有可能产生溢出,使结果出错。

例 2-28 用二进制补码计算 $(+72) - (-98) = (?)$ 。

解

$$(+72)_{10} = (+1001000)_2, \quad (+1001000)_{\text{补}} = 01001000$$

$$(-(-98))_{10} = (+1100010)_2, \quad (+1100010)_{\text{补}} = 01100010$$

0 1 0 0 1 0 0 0 B	+ 72
+ 0 1 1 0 0 0 1 0 B	+ 98
1 0 1 0 1 0 1 0 B	- 86

本例中,一个正数减去一个负数,相当于两个数的绝对值相加。很明显,其结果超出二进制补码的表示范围,因此出现溢出错误。在运算时可从 $C_6 = 1, C_7 = 0$ 而判断得知产生了溢出。

例 2-29 计算 $(-78) - (+86) = (?)$ 。

解 因为 $(-78) - (+86) = (-78) + (-86)$, 所以实际上本例的计算与例 2-27 是完全相同的, 因此结果会产生溢出错误。具体计算过程参考例 2-27。

由上面的讨论可知, 无符号数与有符号数产生溢出的条件因各自可表示数的范围不同而不同。无符号数的溢出判断仅看最高位向前是否有进(借)位, 而有符号数有无溢出产生, 需要看次高位与最高位两位的进(借)位情况。两位都产生进(借)位或都没有产生进(借)位, 则结果无溢出; 否则结果产生溢出。运算时产生溢出, 其结果不正确。计算机对溢出的处理, 一般是产生一个自陷中断, 通知用户采取某种措施。

2.4 定点数与浮点数

在实际生活中的数据除了有正、负, 还带有小数, 在计算机中就要处理含有小数部分的数值数据。首要的问题是小数点位置如何表示。根据小数点位置是否固定, 数的格式分为定点数表示与浮点数表示两种。

2.4.1 定点数

机器数中只有“0”和“1”两种符号, 显然, 并不具有专门的物理设备来表示小数点, 也就是说, 小数点在机器中实际不存在。所谓定点数是将小数点的位置约定在固定的位置上。在计算机中小数点位置固定不变的数称为定点数。为了运算方便, 通常只采取两种简单的小数点位置约定: 把小数点的位置约定在数的最高有效位的左边或最低有效位的右边。即把所有的数化为纯小数或纯整数。

约定数的小数点位置在符号位之后, 在最高数值位之前, 定点数是纯小数, 其表示格式如图 2-7 所示。

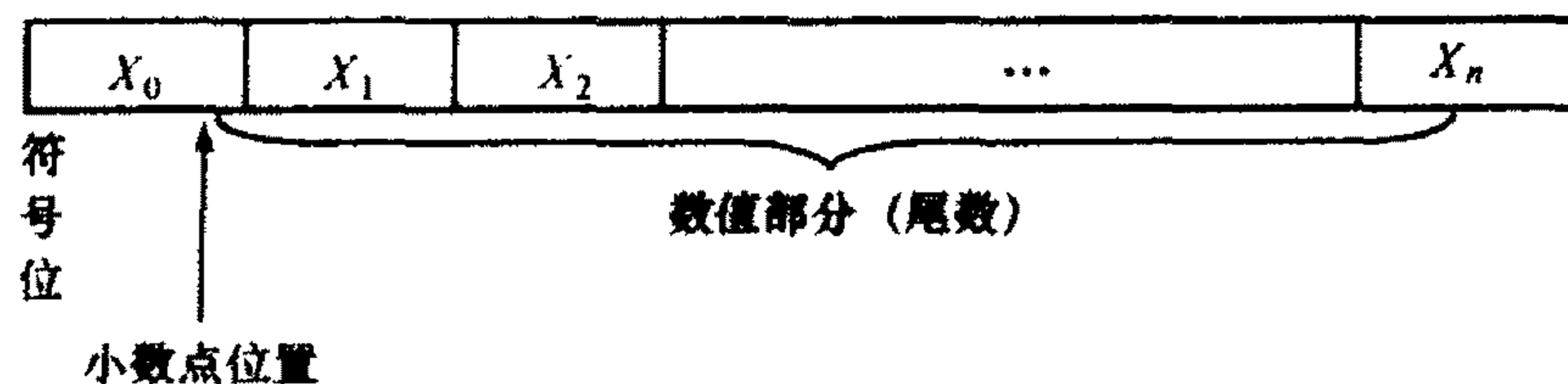


图 2.7 定点小数格式

对于 $n+1$ 位(含符号位)机器字长的定点小数, 不同的机器数表示的范围有所不同:

① 原码定点小数表示范围 $(1-2^n) \sim (1-2^n)$;

② 补码定点小数表示范围 $1 \sim (1 - 2^n)$;

③ 分辨率(即非零绝对值最小数) 2^n 。

对于绝对值小于定点小数分辨率 2^n 的数,通常当作 0 处理。

【例 2-30】 机器字长 $n+1=16$ 的定点小数, $n=15$, 则

① 原码表示范围: $(1 - 2^{-15}) \sim (1 - 2^{-15})$;

② 补码表示范围: $1 \sim (1 - 2^{-15})$ 。

约定数的小数点在最低位的右边,参与运算的数是纯整数。

定点小数表示法主要用在早期的计算机中,以图节省硬件。目前较高性能的计算机都已能处理计算多种类型数据。在此通过定点小数讨论数值数据的不同编码方案,另外,定点小数也被用来表示浮点数的尾数部分。

2.4.2 浮点数

在实际应用中常常会遇到以下格式的十进制实数:

$$492.475\ 9 = 0.492\ 475\ 9 \times 10^3$$

该数既有整数部分也有小数部分,不能用定点数格式直接表示。还可能有如下格式的数:

$$0.000\ 000\ 003 = 0.3 \times 10^{-8}$$

该数的位数过长,可能超出了定点数的表示范围。

以上两个等式右边采用了科学计数法来表示数,在计算机中也引入了类似于科学计数法的方式来表示实数,称为浮点数表示法,即小数点的位置可以左右移动,浮点数包括有符号整数和小数。

在计算机系统中,典型的浮点数格式如图 2-8 所示。

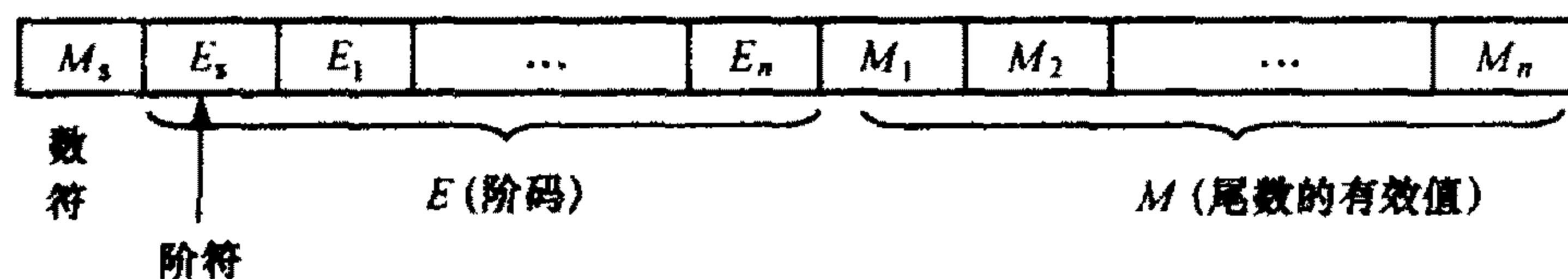


图 2.8 典型浮点格式

图中,浮点数的代码由两部分组成:阶码 E 和尾数 M 。浮点数的真值为

$$N = \pm R^E \times M \quad (2.9)$$

R (Radix)是阶码的底。在机器中一般规定 R 为 2、8 或 16,与尾数的基数相同。例如尾数为二进制,则 R 也为 2。同一种机器的 R 值是固定不变的,所以不需在浮点代码中表示出来,而是隐含约定的。因此,机器中的浮点数只需表示出阶码和尾数部分。

E (Exponent) 是阶码, 即是指数值, 为带符号整数。

E_s 是阶码的符号, 即指数的符号, 决定浮点数范围的大小。

M 是尾数, 或称有效数字, 通常是纯小数, 常用原码或补码表示。

M_s 是尾数的符号位, 安排在最高位。它也是整个浮点数的符号位, 表示该浮点数的正负。

浮点数的表示范围主要由阶码决定, 精度则主要由尾数决定。若不对浮点数的表示做出明确规定, 同一个浮点数的表示就不是惟一的。为了便于运算和比较, 通常采用浮点数规格化形式, 即将尾数的绝对值限定在某个范围之内。如果阶码的底为 2 (即尾数采用二进制), 则规格化浮点数的尾数应满足条件:

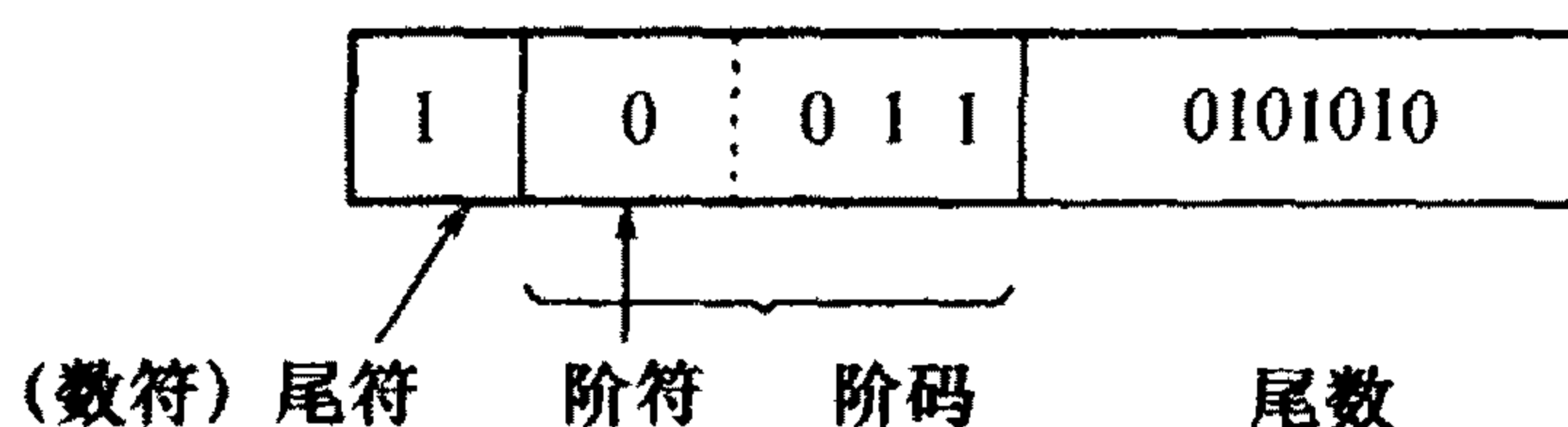
$\frac{1}{2} \leq M < 1$ 。尾数作为定点小数, 其绝对值应小于 1; 由于利用了最高数位, 其绝对值应

大于或等于 $(0.1)_2$, 即 $\frac{1}{2}$ 。从形式上看, 对于正数, 规格化尾数最高数位 $m_1 = 1$, 这表明尾数的有效位数被充分利用了。对于负数补码, 一般情况下尾数最高数位 $m_1 = 0$, 但有一种特殊情况除外, 即 $M = -\frac{1}{2}$ (此时 $m_1 = 1$)。

【例 2-31】某浮点数长 12 位, 其中阶码 4 位, 用补码表示; 数符 1 位, 尾数 7 位, 用补码表示。写出二进制数 $(-101.011)_2$ 的规格化浮点代码。

$$(-101.011)_2 = (-0.101011)_2 \times 2^{+3}$$

则其浮点代码为:



例 2-32 例 2-31 浮点格式设有浮点代码为 0,0011,0101010, 尾数最高数位 m_1 未充分利用, 仍是一个未规格化的浮点数。可将尾数左移 1 位, 同时阶码减 1, 浮点代码则调整为规格化形式 0,0010,1010100。

在浮点表示法中, 小数点的位置是由阶码 (包括阶符) 决定的, 改变阶码可以改变小数点的位置。

2.5 二进制编码

计算机只能处理二进制数据, 但人们更习惯于用十进制数, 所以在某些情况下也希望计算机能直接处理十进制形式表示的数据。此外, 现代计算机不仅要处理数值领域的问题, 而

且被大量地用于处理非数值领域的问题,如文字处理、信息发布、数据库系统等,这就要求计算机还应能识别和处理文字、字符和各种符号,如:

- ① 数字 0,1,⋯,9;
- ② 字母 26 个大小写的英文字母: A,B,⋯,Z,a,b,⋯,z;
- ③ 专用符号 +, -, *, /, ↑, \$, %⋯;
- ④ 控制字符 CR(回车),LF(换行),BEL(响铃)⋯。

所有这些字符、符号以及十进制数最终都必须转换为二进制格式的代码才能为计算机所处理,即字符和十进制数都要表示为若干位二进制码的组合,即信息和数据的二进制编码。

1. 二进制编码的十进制数

用二进制编码表示的十进制数,称为二—十进制码(Binary Coded Decimal, BCD)码。它的特点是遵循“逢十进一”的法则,或者说保留了十进制的权,而数字则用 0 和 1 的组合编码来表示。BCD 码的编码方案有多种,下面只介绍最常用的一种 BCD 码,即 8421 码。

1) 8421 码

8421 BCD 码(简称 BCD 码)用 4 位二进制编码表示 1 位十进制数,其 4 位二进制编码的每一位都有特定的位权,从左至右分别为: $2^3 = 8$, $2^2 = 4$, $2^1 = 2$, $2^0 = 1$,故称其为 8421 码。

BCD 码与二进制码的对应关系见表 2-8。

表 2-8 BCD 码与二进制码的对应关系

十进制数	8421 码
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

十进制数只有 0~9 十种状态,而 4 位二进制数可表示十六种状态(0000~1111),所以 BCD 码只使用了 0000~1001 来表示十进制数的 0~9,其余的六种状态 1010~1111 在 BCD 码中是非法编码。

2) BCD 码与十进制数、二进制数的转换

一个十进制数用 BCD 码来表示是非常简单的, 只要对十进制数的每一位按表 2-8 的对应关系单独进行转换即可。

例 2-33 将十进制数 234.15 用 BCD 码表示。

解

$$(234.15)_{10} = (0010\ 0011\ 0100.0001\ 0101)_{\text{BCD}}$$

同样, 也能很容易地由 BCD 码得出其对应的十进制数。如 BCD 码

$$0110\ 0011\ 1001\ 1000.0101\ 0010$$

对应的十进制数为 6398.52。

BCD 码与二进制数之间的转换要稍微麻烦一些, 一般需要借助于十进制数作为中间桥梁进行转换。

例 2-34 将 BCD 码 $(0001\ 0001.0010\ 0101)_{\text{BCD}}$ 转换为二进制数。

解

$$(0001\ 0001.0010\ 0101)_{\text{BCD}} = (11.25)_{10}$$

$$(11.25)_{10} = (1011.01)_2$$

所以

$$(0001\ 0001.0010\ 0101)_{\text{BCD}} = (1011.01)_2$$

例 2-35 将二进制数 01000111 转换为 BCD 码。

解

$$(01000111)_2 = (71)_{10}$$

所以

$$(01000111)_2 = (0111\ 0001)_{\text{BCD}}$$

为了避免与二进制数混淆, 在书写 BCD 码时, 通常将每位 BCD 码(4 位二进制编码)写成一组, 中间加一个空格, 且要加上 BCD 下标。

3) 计算机中 BCD 码的存储方式

计算机的存储单元通常以字节(8 个二进制位)为最小单元, 很多操作也是以字节为单位进行的, 在一个字节中如何存放 BCD 码有两种方式, 即压缩的 BCD 码和非压缩的 BCD 码。因为一个 BCD 码有 4 个二进制位, 所以可以在一个字节中存放 2 个 BCD 码, 这种方式称为压缩 BCD 码表示法。在采用压缩 BCD 码表示十进制数时, 一个字节表示两位十进制数。例如 10010010B 表示十进制数 92。

非压缩的 BCD 码(又称扩展 BCD 码)表示法是每个字节只存放一个 BCD 码, 即低 4 位为 BCD 码, 高 4 位全为 0。例如, 同样是十进制数 92, 用非压缩 BCD 码表示为 00001001

00000010。

2. 字符的编码

各种字符和符号也必须按特定的规则用二进制编码才能在机器中表示。目前在微型计算机中普遍采用的字符编码系统是 ASCII 码 (American Standard Code for Information Interchange, 美国国家标准信息交换码), ASCII 字符编码表参见附录 A。它用 7 位二进制编码来表示 128 个字符和符号。

微型计算机中一个字节为 8 位, 一般规定一个 ASCII 码存放在字节的低 7 位, 字节最高位 D_7 位恒为 0 (但在通信中常把最高位用做奇偶校验位)。这样, 用一个字节来表示一个 ASCII 字符编码: 数字 0 ~ 9 的 ASCII 码为 30H ~ 39H; 26 个英文大写字母 A ~ Z 的 ASCII 码为 41H ~ 5AH; 26 个英文小写字母 a ~ z 的 ASCII 码为 61H ~ 7AH。

有时在 7 位 ASCII 码的最高位之前加上 1 位做奇偶校验位, 用来指明数据传送过程中是否有 1 位出现了错误。

奇偶校验分为奇校验和偶校验。偶校验的含义是, 包括这 1 位在内的 8 位二进制码中为 1 的位数之和为偶数。例如, 字母 A 的 ASCII 码为 1000001B。奇校验的含义是, 包括校验位在内, 所有为 1 的位数之和为奇数。根据这个原则, 具有奇校验位的 A 的 ASCII 码就是: 11000001B。

习 题 二

2.1 计算机中常用的计数制有哪些?

2.2 8 位二进制数原码可表示数的范围是(); 8 位二进制数补码可表示数的范围是(); 8 位二进制数反码可表示数的范围是()。

2.3 16 位二进制数的原码、补码、反码可表示的数的范围是多少?

2.4 完成下列数制的转换:

(1) $10100110B = ()D = ()H$

(2) $0.11011 = ()D$

(3) $253.398 = ()B$

(4) $1011011.101B = ()H = ()_{BCD}$

2.5 设字长 8 位 (含 1 位数符), 分别写出下列各二进制数的原码、补码和反码:

(1) 0 (2) -0 (3) 0.001

(4) -0.1101 (5) 1101 (6) -1101

2.6 写出下列真值对应的原码和补码的形式:

(1) $X = -1110011B$

(2) $X = -71D$

(3) $X = +1001001B$

2.7 将下列十进制数转换为二进制数:

(1) 128 (2) 0.34

(3) 65.23 (4) $11/256$

2.8 变换以下十六进制数到 BCD 码:

(1) 23H (2) 0AD4

(3) 34.AD (4) BD32

2.9 写出下列字符的 ASCII 码:

A 9 * = !

2.10 若加上奇校验,以下字符的 ASCII 码是什么?

B 4 7 = +

2.11 设阶码 8 位(用补码表示),尾数 23 位,数符 1 位。若浮点数 X 的十六进制存储格式为 $(41390000)_{16}$,求其 32 位浮点数的十进制值。

2.12 二进制浮点数补码表示为 49AH,前 4 位为阶码,后 8 位为尾数,符号位均为 1 位,问真值十进制数为多少?

第3章 微处理器

本章概述微处理器的一般结构,较为详细地讨论 8086 微处理器的外部引出线、内部结构及内部寄存器组。另外,还用一定篇幅介绍部分新型 CPU 的工作原理、内部结构及其新技术。

3.1 微处理器的一般结构

微处理器(CPU)是计算机系统的核心部件,控制和协调整个计算机系统的工作,具有以下几项基本功能:

- ① 能够进行算术运算和逻辑运算;
- ② 能对指令进行译码、寄存,并执行指令所规定的操作;
- ③ 具有与存储器和 I/O 接口进行数据通信的能力;
- ④ 具有少量数据的暂存能力;
- ⑤ 能够提供这个系统所需的定时和控制信号;
- ⑥ 能够响应输入/输出设备发出的中断请求。

人们通常用 CPU 内部操作中的数据位数(内部寄存器位数及内部数据总线位数)来作为对其总体性能的一个表征。通常所说的 16 位机、32 位机是表示该计算机中微处理器内部数据总线的宽度,即 CPU 可同时操作的二进制码的位数。微型计算机有 8 位、16 位或 32 位 CPU 等,其含义是可操作 8 位、16 位或 32 位二进制码。目前常用的 CPU 都是 32 位的,即一次可传送 32 位二进制数。

微处理器的发展速度很快,第一代微处理器是在 1971 年由 Intel 公司生产的,型号为 4004(4 位 CPU)和 8008(8 位 CPU)。这之后的 30 多年中,它经历了六代产品的更新换代,在速度、功能等各方面都有了很大的提高。表 3-1 列出了 Intel 公司从最初的第一代产品到目前的第六代 CPU 的一些主要技术参数。

微处理器的内部由 3 个部分组成,即运算器、控制器和寄存器组,寄存器组又可视为运算器部件中的一部分。下面分别来看一下它们的组成及功能。

表 3-1 Intel 主要 CPU 芯片一览表

代	出品年份	字长/B	型 号	线宽/ μm	晶体管数/万个	时钟频率/MHz	取指速度/MIPS
一	1971	4	4004	50	0.2	< 1	0.05
	1972	8	8008		0.3		
二	1974	8	8080	20	0.5	2 ~ 4	0.5
三	1978	16	8086	2 ~ 3	2.9	4.77 ~ 10	< 1
	1982		80286		13.4	8 ~ 16	1 ~ 2
四	1985	32	80386	1 ~ 2	27.5	16 ~ 33	6 ~ 12
	1989		80486		120	25 ~ 66	20 ~ 40
五	1993	32	Pentium	0.6 ~ 0.8	330	60 ~ 200	100 ~ 200
六	1995	32	Pentium Pro	0.6	550	133 ~ 200	> 300
	1996		Pentium MMX	0.6	450	166 ~ 233	
	1997		Pentium II	0.35	750	233 ~ 450	
	1999		Pentium III	0.25 ~ 0.13	850	450 ~ 1 200	
	2002		Pentium 4	0.18 ~ 0.13	1 000	1 300 ~ 3 060	
	2002	64	Itanium	0.13	CPU: 25 000 Cache: 30 000	800(20 条指令/时钟周期)	> 3 000

注：Itanium 具有 64 位数据总线，但仅有 32 位地址总线，所以仍称它为 32 位的微处理器。

3.1.1 运算器

运算器由算术逻辑单元(Arithmetic Logical Unit, ALU)、通用或专用寄存器组及内部总线三个部分组成。其核心功能是实现数据的算术运算和逻辑运算，所以有时也将运算器称为算术逻辑运算单元。

ALU 的内部包括负责加、减、乘、除运算的加法器，以及实现与、或、非、异或等逻辑运算的逻辑运算功能部件。1 位算术逻辑运算单元的结构如图 3.1 所示。

除了作为核心部件的 ALU 外，运算器还具有提供操作数和暂存中间运算结果及结果特征的寄存器组以及数据传送通道。在 CPU 内部用于传送数据和指令的传送通道称为 CPU 内部总线。运算器的结构根据其内部总线数量的不同分为三种，如图 3.2 所示。

1) 单总线结构运算器

图 3.2(a)是单总线结构运算器的示意图，此时所有的

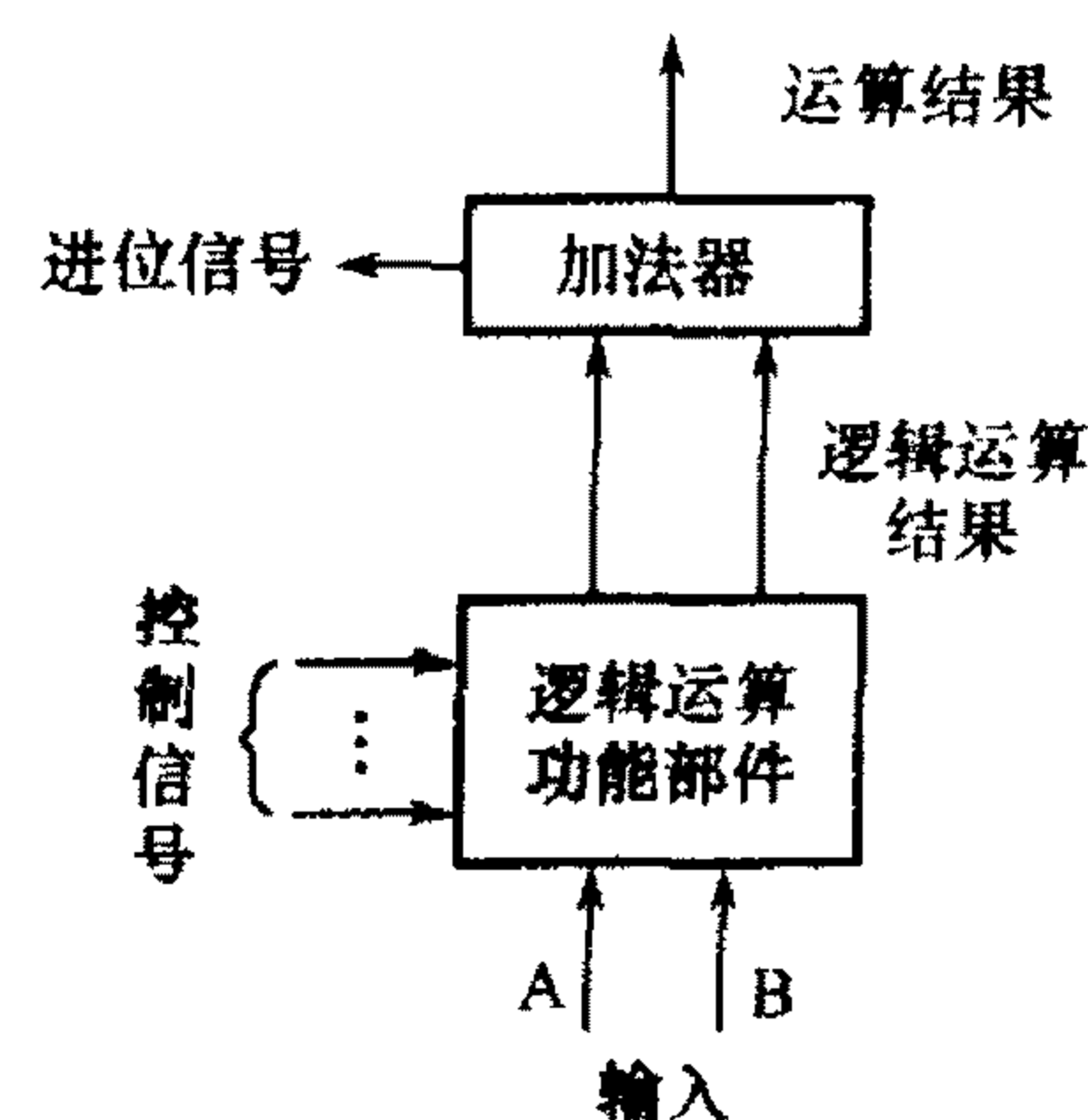


图 3.1 1 位算术逻辑运算单元结构示意图

部件都通过一条内部总线传递信息,任何时刻只有一组数据从源部件传送到目标部件。从图中可看出,ALU 的输入端有两个用来暂时存放参加运算的操作数的锁存器。当要进行一次双操作数的运算时,首先通过总线将第一个操作数放入锁存器 A 或 B 中,然后通过总线传送另一个操作数至另一个锁存器,再进入 ALU 进行运算,运算的结果通过总线置入某个内部通用寄存器。这种结构的控制简单,但速度比较慢。

2) 双总线结构运算器

双总线结构是在运算器内部用两条总线来传送操作数,如图 3.2(b)所示。此时参加运算的两个操作数可同时通过两条总线送至 ALU 进行运算,运算的结果经缓冲器再通过任意一条总线传送到通用寄存器。这种结构的运算器处理速度显然就要比单总线结构的快。

3) 三总线结构运算器

速度最快的运算器结构是图 3.2(c)所示的三总线结构。它用两条总线来传送操作数,一条专门用于传送运算结果。这样,在传送运算结果的同时就可通过另外两条总线传送参加运算的操作数,只要 ALU 速度足够快,全部操作就可一步完成。

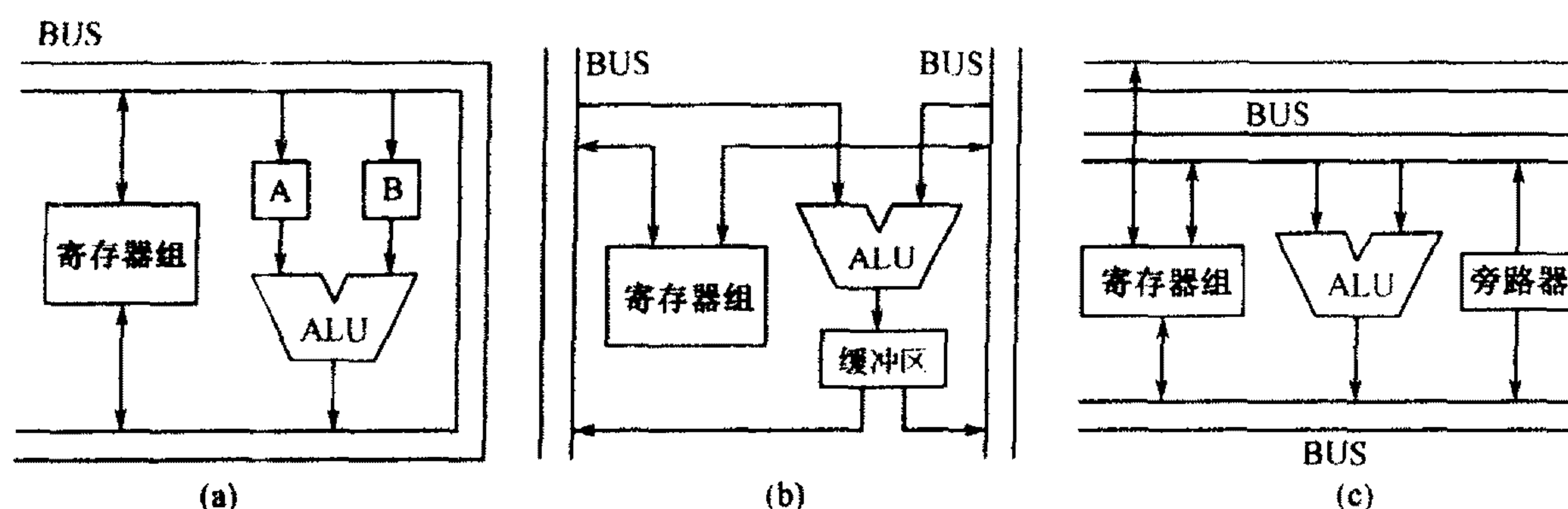


图 3.2 运算器结构示意图

3.1.2 控制器

控制器的作用是控制程序的执行,是整个系统的指挥中心,必须具备以下几项基本功能:

1) 指令控制

计算机的工作过程就是连续执行指令的过程。指令在存储器中是连续存放的,一般情况下,指令被按照顺序一条条地取出执行,只有在碰到转移类指令时才会改变顺序。控制器要能根据指令所在的地址按顺序或在遇到转移指令时按照转移地址取出指令,分析指令(指令译码),传送必要的操作数,并在指令执行结束后存放运算结果。总之,要保证计算机中的指令流的正常工作。

2) 时序控制

指令的执行是在时钟信号的严格控制下进行的,一条指令的执行过程称为指令周期,不同指令的指令周期中所包含的机器周期数是不相同的,而一个机器周期有多少个时钟周期也不一定一样。这些时序信号用于计算机的定时,它们由控制器产生,使系统按一定的时序关系进行工作。

3) 操作控制

根据指令流程,确定在指令周期的各个节拍中要产生的微操作控制信号,以有效地完成各条指令的操作过程。

除此之外,控制器还要具有对异常情况 & 某些外部请求的处理能力,例如出现运算溢出、中断请求等。

控制器的内部主要由以下几个部分组成:

① 程序计数器 PC(Programming Counter) 用来存放下一条要执行的指令在存储器中的地址。在程序执行之前,应将程序的首地址(程序中第一条指令的地址)置入程序计数器。

② 指令寄存器 IR(Instruction Register) 用于存放从存储器中取出的待执行指令。

③ 指令译码器 ID(Instruction Decoder) 指令寄存器中待执行的指令须经过“翻译”才能明白它要执行什么样的操作,这就是指令译码,也就是指令译码器的主要功能。

④ 时序部件 产生计算机工作中所需的各种时序信号。

⑤ 微操作控制部件 这一部分是控制器的主体。一条指令的执行过程可更进一步地看做是一个微操作的产生过程,微操作控制部件用于产生与各条指令相对应的微操作。它根据当前正在执行的指令,在指令的各机器周期的各个节拍内产生相应的微操作控制信号,从而控制整个系统部件的工作。

微处理器中控制器的结构如图 3.3 所示。从图中可以看出,其中的核心部件是微操作控制部件。

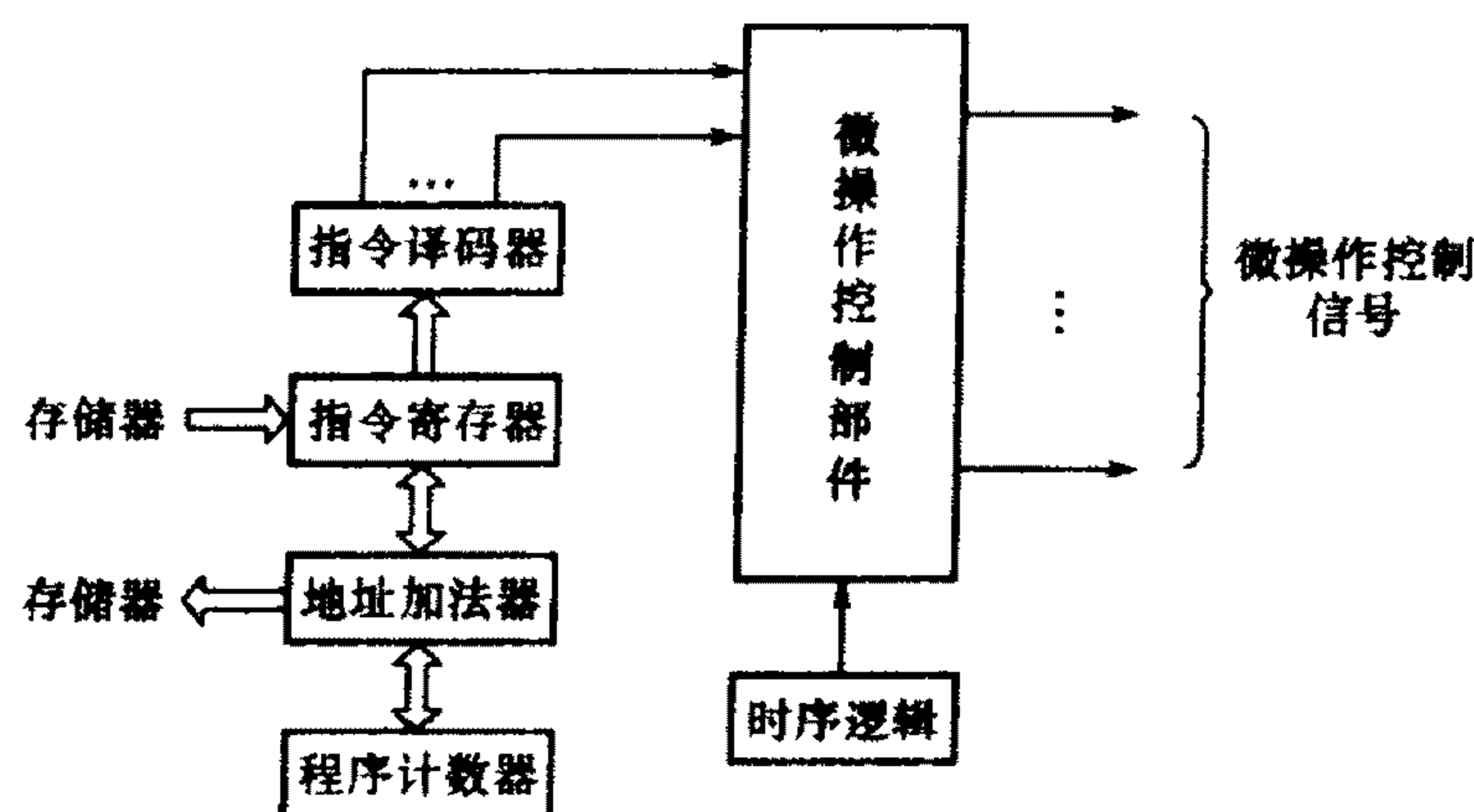


图 3.3 控制器结构示意图

3.2 8086 微处理器

8086 微处理器是典型的 16 位微处理器,即它的内部数据总线、内部寄存器以及外部数据总线都是 16 位的。它是 Intel 公司第三代 CPU,时钟频率为 5 MHz,是具有 40 根引出线的双列直插式芯片,共有 20 位地址线,其中 16 位为数据/地址复用线,可寻址 1 MB 内存单元、64 KB 的 I/O 地址空间。其副产品 8088 CPU 除外部数据总线是 8 位的之外,在其他方面的性能几乎完全一样。

以 8086/8088 为处理器的 IBM-PC 个人计算机曾风靡全世界,也使 8086/8088 成为最主要的微处理器结构。

3.2.1 功能结构及其特点

1. 结构特点

与 8080 相比,8086/8088 在结构上有了如下的几项改进,从而使它在性能上有了很大的提高。

1) 指令流水线

8086/8088 与 8080 的最重要的区别是采用了流水线技术。尽管在这方面还不能与现在新型的 CPU(如 Pentium、K7 等)相比,但由于是它首先在微处理器中采用流水线结构技术的,从而使它成为 CPU 发展史上的一个里程碑。

在程序执行过程中,CPU 有规律地重复执行以下步骤:

- ① 从存储器中取一条指令;
- ② 指令译码;
- ③ 读取操作数(如果需要);
- ④ 执行指令;
- ⑤ 存放结果(如果必要)。

这里,读取操作数和存放结果不是必需的操作,要根据指令的情况来定。必需的操作是取指令、指令译码和指令执行 3 步。其中,取指令由 CPU 内部的总线接口单元(Bus Interface Unit, BIU)完成,而指令译码和执行由执行单元(Execution Unit, EU)实现。8088/8086 CPU 以前的微处理器都是用串行方式完成这些操作的,即依照取指令、译码、执行指令这样的顺序一步一步地执行。CPU 的取指令和执行指令不能并行进行;另外,某些指令的执行并不需要访问内存,所以在指令的执行过程中外部总线也处于空闲状态。工作过程如图 3.4(a)所

示。这样的结构显然使 CPU 和总线的工作效率都比较低。

从 8086/8088 开始,CPU 采用了一种新的结构来并行地完成这些工作。执行单元负责执行指令,总线接口单元负责取指令、取操作数和写结果。它们独立地、并行地完成各自相应的工作。当 EI 执行指令时,BIU 便“预取”下一条要执行的指令,所以大多数情况下取指令的时间可“省掉”,从而加快了程序的运行速度。并行操作的示意图如图 3.4(b)所示。

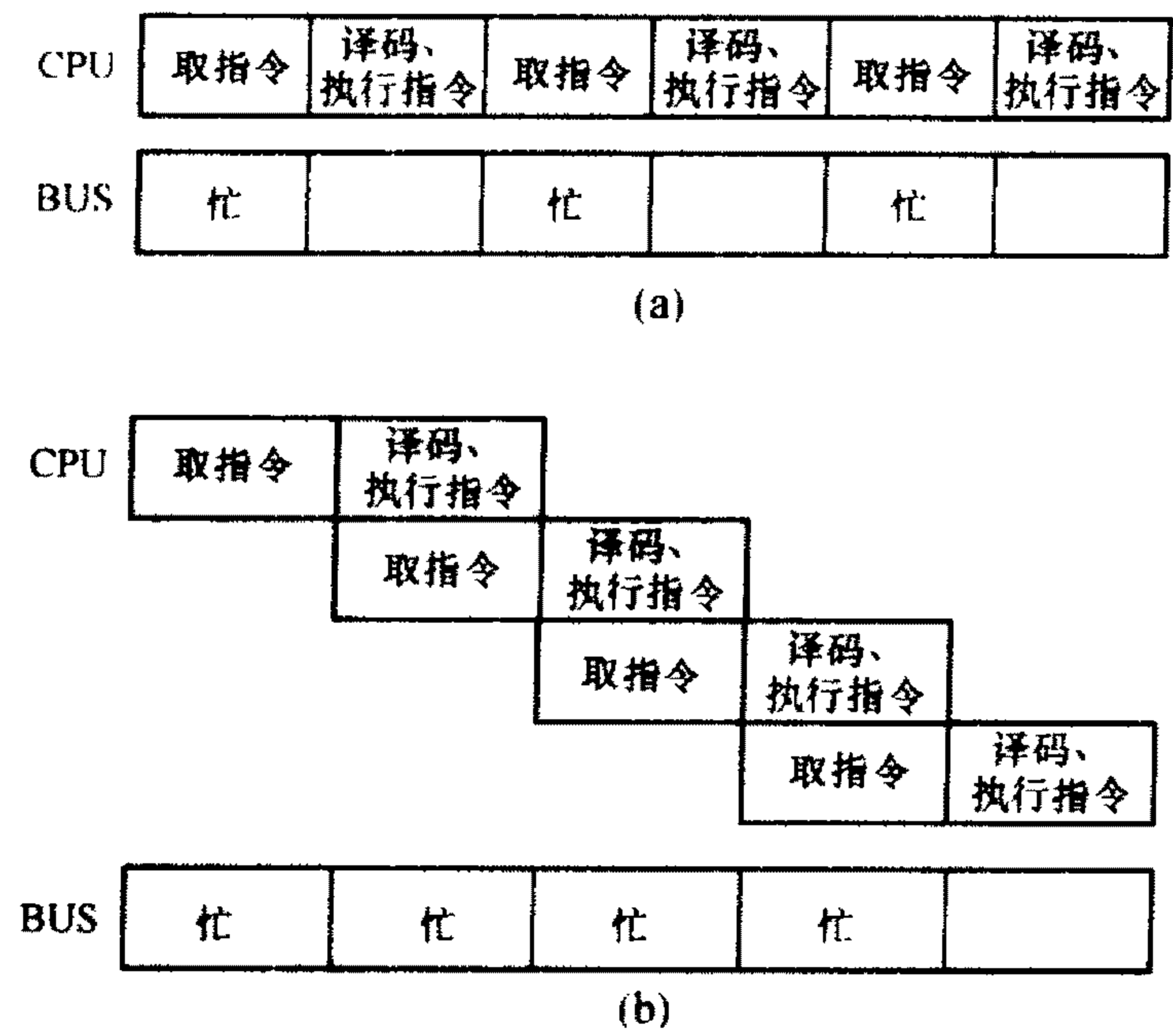


图 3.4 重叠的取指令和执行指令操作(流水线)

这就带来一个问题,因预取的指令有可能不是马上执行,故需要有一个暂时存放所预取的指令的地方。8086 内部设置了一个 6 B(8088 为 4 B)的指令流队列 (Instruction Stream Queue),实际上是一个内部的存储器阵列,存储数据为先进先出。

2) 存储器的分段结构

8086 的地址总线为 20 位,可寻址 $2^{20} = 1\text{ MB}$ 的内存空间;但其内部寄存器和内部地址总线都只有 16 位,也就是说能够由 ALU 提供的最大地址空间只能是 64 KB。为了实现 CPU 对 1 MB 空间的寻址,人们将内存储器空间分为若干逻辑段,每个段最大为 64 KB,并在 CPU 中专门设置了一些段寄存器,用于存放逻辑段的起始地址,这些起始地址是 16 位的,满足内部地址总线的宽度要求。

一个实际的存储器单元的地址是由它所在段的段地址和该单元与段起始地址之间的位移量(偏移地址)按一定规律共同构成的一个 20 位的地址,称为物理地址。通过这样的方法,使得 8086 能够实现对 1 MB 空间的管理。有关物理地址的构成方法将在 3.4 节介绍。

3) 支持用于浮点运算的协处理器及多微处理器系统

在 8086 以前的微处理器中,都是采用定点数据表示来实现浮点运算的,实现的方法大多是通过子程序来实现。这样做很费时,一般要使处理器的速度降低两个数量级。如用一台定点运算速度为 1 000 万次/s 的计算机进行科学计算,则其实际运算速度将低于 10 万次/s。不仅如此,CPU 与主存储器之间的数据通信量也会大大增加。

8086/8088 CPU 接用于浮点运算的数学协处理器 8087,可大大地提高系统的运算速度和数据处理能力。

另外,8086/8088 CPU 在指令方面和结构设计上,都考虑了支持使用该微处理器构成一个共享总线的多微处理器系统。

2. 功能结构

8086 的内部结构如图 3.5 所示。它由执行单元 EU 和总线接口单元 BIU 两大部分构成。执行单元 EU 负责执行指令。它由算术逻辑单元(ALU)、通用寄存器组、16 位标志寄存器(FLAGS)、EU 控制电路等组成。EU 在工作时直接从指令流队列中取指令代码,对其译码后产生完成指令所需要的控制信息。数据在 ALU 中进行运算,运算结果的特征保留在标志寄存器 FLAGS 中。

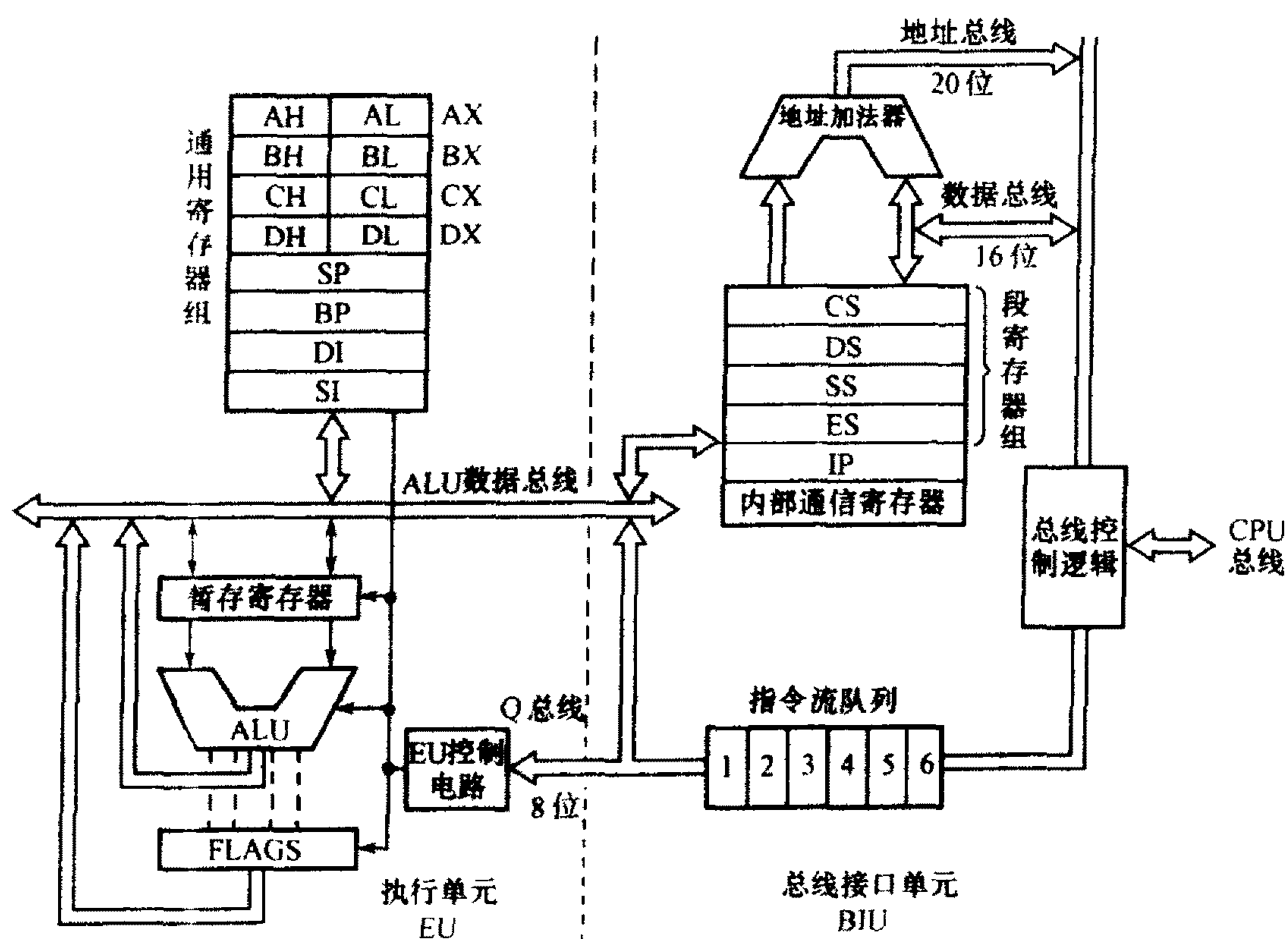


图 3.5 8086 的内部结构

总线接口单元 BIU 负责 CPU 与存储器和 I/O 接口之间的信息传送。它由段寄存器、指令指针寄存器、指令流队列、地址加法器以及总线控制逻辑组成。8088 的指令队列长度为

4 B, 8086 的指令队列长度为 6 B。

当 EU 取走指令、指令流队列出现 2 个以上空单元时, BIU 就自动执行一次取指令周期, 从内存中取出后续的指令代码放入队列中; 如果 EU 需要数据, BIU 会根据 EU 给出的地址, 从指定的内存单元或外设中取出数据供 EU 使用; 运算结束后, BIU 负责将运算结果送入指定的内存单元或外设。如果指令流队列为空, EU 就等待, 直到有指令为止。若 BIU 正在取指令时 EU 发出访问总线的请求, 则必须等 BIU 取指令完毕后, 该请求才能得到响应。一般情况下, 程序是顺序执行的, 而当遇到跳转指令时, BIU 使指令队列复位, 从新地址取出指令, 并立即传给 EU 去执行。

指令流队列的存在使 8086/8088 的 EU 和 BIU 并行工作, 从而减少了 CPU 为取指令而等待的时间, 提高了 CPU 的利用率, 加快了整机的运行速度。另外也降低了对存储器存取速度的要求。

3.2.2 引脚定义及总线结构

1. 8086 的外部引脚及其功能

8086 和 8088 都是具有 40 条引出线的、采用双列直插式封装的集成电路芯片, 其外部引脚如图 3.6 所示。为减少芯片的引脚数, 以减少体积, 8086 的许多引脚都具有双重定义和功能, 采用分时复用方式工作, 即在不同时刻, 引脚上的信号具有不同的意义。引脚定义的方法大致可分为这样几种:

- ① 每根引脚只传送一种信息, 如读信号线 \overline{RD} 。
- ② 引脚电平的高低代表不同的信号, 如 M/\overline{IO} , 在为低电平时, 表示当前访问的是存储器; 为高电平时, 则表示访问输入/输出接口。
- ③ 不同的时间引脚传送不同的信息, 即分时复用。如 $AD_0 \sim AD_{15}$, 在某一时刻传送地址的低 16 位信号; 另一时刻则传送 16 位数据。
- ④ 在引脚作为输入端或输出端时传送不同的信息。如 $\overline{RQ}/\overline{GT}_0$ 端, 作为输入端时, 输入的是总线请求信号; 为输出端时, 则输出总线请求允许信号。
- ⑤ 当 CPU 工作在不同模式时, 引脚具有不同的名称和定义。8086 微处理器有两种工作模式, 最大模式和最小模式。两种工作模式下的引脚定义有一些区别, 图 3.6 所示右边括号中的引脚名称就是 CPU 工作在最大模式时对应引脚的含义。如第 29 根引脚, 在最小模式下的定义是写信号 \overline{WR} , 而在最大模式下的定义则为总线封锁信号 \overline{LOCK} 。8086 在两种工作模式下公用引脚的含义如下:

1) 地址/数据总线

8086 有 20 位地址线, 16 位数据线, 采用分时复用方式, 共同占用 20 根引脚。

$AD_0 \sim AD_{15}$ 地址、数据分时复用的双向信号线,三态。当 $ALE = 1$ 时,这些引脚上传输的是地址信号;当 $\overline{DEN} = 0$ 时,这些引脚上传输的是数据信号。

$A_{16} \sim A_{19}/S_3 \sim S_6$ 分时复用的地址/状态信号线,三态输出。在 8086 访问存储器时,读/写总线周期的第一个机器周期 T_1 从这 4 个引脚上送出最高 4 位地址 $AD_{16} \sim AD_{19}$ 。而在总线周期的其他机器周期,这 4 个引脚送出状态信号 $S_3 \sim S_6$ 。这些状态信号里, S_6 始终为低电平; S_5 指示标志寄存器的中断允许标志位 IF 的状态; S_4 、 S_3 的组合指示 CPU 当前正在使用的段寄存器,其编码见表 3-2。

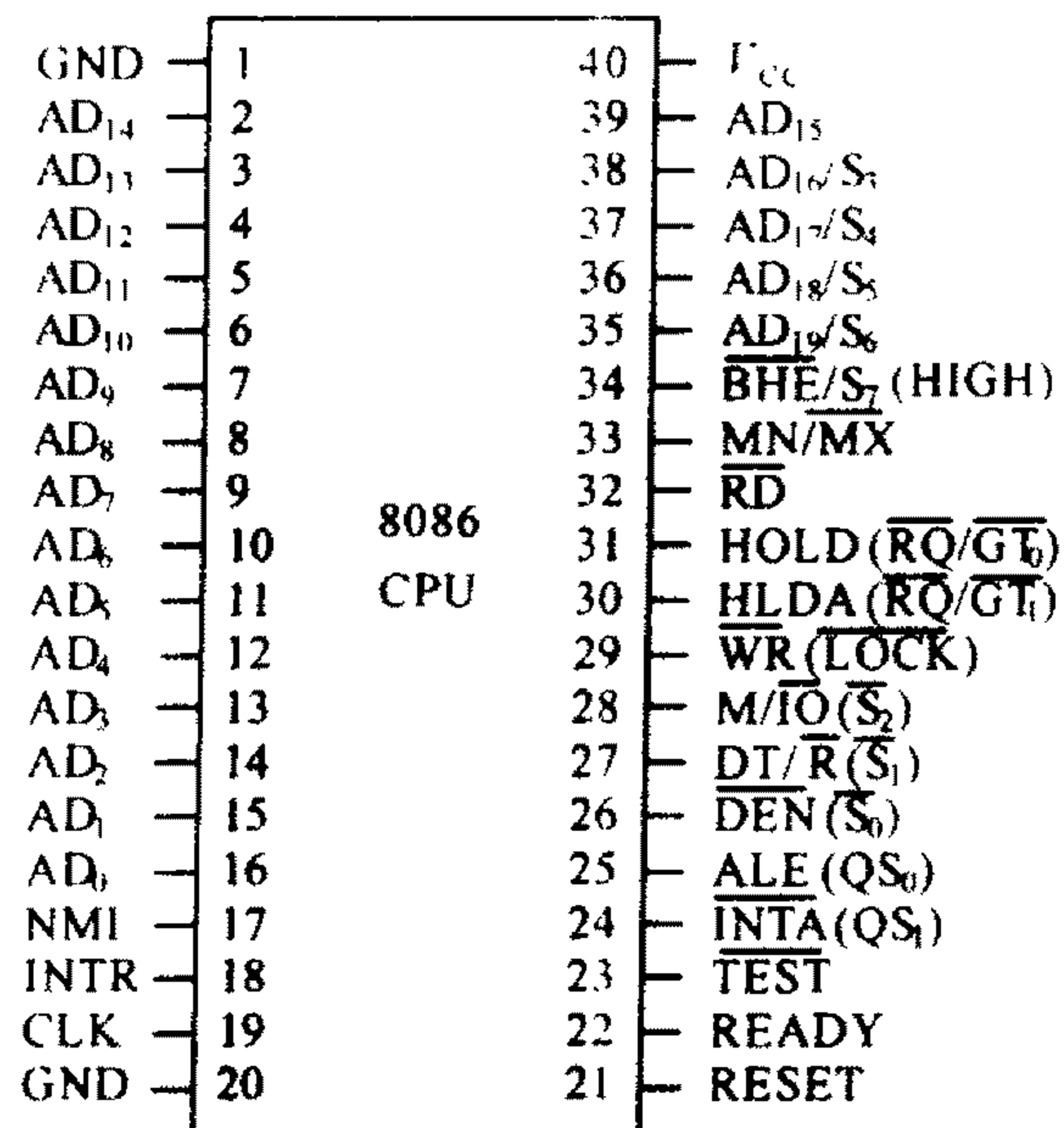


图 3.6 8086 微处理器芯片引脚图

表 3-2 S_3 和 S_4 的功能

S_4	S_3	当前正在使用的段寄存器
0	0	ES
0	1	SS
1	0	CS 或未使用任何段寄存器
1	1	DS

当 CPU 访问 I/O 端口时,这 4 根引脚不使用,全部为低电平。

2) 控制总线

控制总线引脚共有 16 根,这里先讨论两种工作模式共用的 8 根引脚。

$\overline{MN}/\overline{MX}$ 工作方式控制输入。为高电平时,表示 CPU 工作在最小模式;为低电平时,表示 CPU 处于最大模式。

\overline{RD} 读选通信号,三态,低电平有效。当其有效时,表示 CPU 正在对存储器或 I/O 接口进行读操作。

READY “准备好”信号输入引脚,高电平有效。它是由被访问的内存或 I/O 设备发出的响应信号,当其有效时,表示存储器或 I/O 设备已准备好,CPU 可以进行数据传送。

若存储器或 I/O 设备未准备好,则 READY 信号为低电平。CPU 在 T_3 周期采样 READY 信号,若其为低电平,CPU 自动插入等待周期 T_w (1 个或多个),直到 READY 变为高电平后,CPU 才脱离等待状态,完成数据传送过程。

INTR 可屏蔽中断请求输入信号,高电平有效。CPU 在每条指令的最后一个周期采样该信号,以决定是否进入中断响应周期。这个引脚上的中断请求信号可用软件屏蔽。

\overline{TEST} 测试信号输入引脚,低电平有效。当 CPU 执行 WAIT 指令时,每隔 5 个时钟周

期对此引脚进行一次测试。若为高电平, CPU 则继续处于空转状态进行等待, 直到 $\overline{\text{TEST}}$ 引脚变为低电平后, CPU 才结束等待状态, 继续执行下一条指令。

$\overline{\text{NMI}}$ 非屏蔽中断请求输入信号, 上升沿触发。这个引脚上的中断请求信号不能用软件屏蔽, CPU 在当前指令执行结束后就进入中断过程。

$\overline{\text{RESET}}$ 系统复位输入信号, 高电平有效。为使 CPU 完成内部复位过程, 该信号至少要在 4 个时钟周期内保持有效。复位后 CPU 内部寄存器的状态见表 3-2。当 RESET 返回低电平时, CPU 将重新启动。

$\overline{\text{BHE}}/\text{S}_7$ 分时复用的控制/状态信号线, 三态输出。在总线周期的第一个时钟周期输出 $\overline{\text{BHE}}$ 信号, 其他时钟周期输出状态信号 S_7 (S_7 的意义目前没有定义)。 $\overline{\text{BHE}}$ 信号的意义是: 当 $\overline{\text{BHE}}$ 为低电平时, 表示可使用高 8 位数据线 $\text{AD}_8 \sim \text{AD}_{15}$; 否则只使用低 8 位数据线 $\text{AD}_0 \sim \text{AD}_7$ 。

$\overline{\text{BHE}}$ 信号和地址信号一样需要锁存, 它同最低位地址信号 A_0 的状态组合在一起表示的功能见表 3-3。

表 3-3 $\overline{\text{BHE}}$ 与地址信号 A_0 的状态组合功能

操 作	$\overline{\text{BHE}}$	A_0	使用的数据线
读或写偶地址的一个字	0	0	$\text{AD}_0 \sim \text{AD}_{15}$
读或写偶地址的一个字节	1	0	$\text{AD}_0 \sim \text{AD}_7$
读或写奇地址的一个字	0	1	$\text{AD}_8 \sim \text{AD}_{15}$
读或写奇地址的一个字节	0	1	$\text{AD}_0 \sim \text{AD}_7$ (第一个总线周期放入低 8 位数据)
	1	0	$\text{AD}_8 \sim \text{AD}_{15}$ (第二个总线周期放入高 8 位数据)

2. 8086 的两种工作模式及其引脚定义

8086 具有两种工作模式: 最小模式和最大模式。最小模式又称为单微处理器模式, 在这种模式下, CPU 仅支持由少量设备组成的单微处理器系统而不支持多处理器结构, 小系统所需要的全部控制信号都由 CPU 直接提供。对应地, 最大模式又称为多微处理器模式。在最大模式下, 系统中除了有 8086 CPU 之外, 还可以接另外的处理器 (如 8087 数学协处理器), 构成多微处理器系统。此时 CPU 不直接提供读/写命令等控制信号, 而是将当前要执行的传送操作类型编码成 3 个状态位输出, 由总线控制器对状态信号进行译码后产生相应控制信号。其他的控制引脚则直接提供最大模式系统所需要的控制信号。

两种工作模式可以通过在 $\text{MN}/\overline{\text{MX}}$ 输入引脚加上不同的电平来进行选择。当 $\text{MN}/\overline{\text{MX}} =$

1 时,8086 工作在最小模式;当 $\overline{MN}/\overline{MX} = 0$ 时,8086 工作在最大模式。两种工作模式下的部分引脚具有不同的功能。

1) 最小模式下的引线

引脚 24 ~ 31 在最小模式下的功能定义为:

\overline{INTA} 中断响应输出端。当 CPU 响应从 INTR 端输入的中断请求时,由 \overline{INTA} 端输出两个连续的负脉冲,可用做外部中断源的中断向量码的读选通信号。

ALE 地址锁存允许信号,三态输出,高电平有效。当它为高电平时,表明 CPU 地址线上有有效地址。可利用它的下降沿将地址信号 $A_0 \sim A_{19}$ 和 \overline{BHE} 信号锁存到地址锁存器中。

\overline{DEN} 数据允许信号,三态,低电平有效。该信号有效时,表示数据总线上有有效数据。它在每次访问内存或 I/O 接口以及在中断响应期间有效。它常用做数据总线驱动器的片选信号。

DT/\overline{R} 数据传送方向控制信号,三态。用于确定数据传送的方向。高电平时,CPU 向存储器或 I/O 端口发送数据;低电平时,CPU 从存储器或 I/O 接口接收数据。此信号用于控制总线收发器的传送方向。

M/\overline{IO} 输入/输出/存储器控制信号,三态。用来区分当前操作是访问存储器还是访问 I/O 端口。引脚输出为高电平时,表示访问存储器;为低电平时,则表示访问 I/O 端口。

\overline{WR} 写信号输出,三态。此引脚输出为低电平时,表示 CPU 正在对存储器或 I/O 端口进行写操作。

HOLD 总线保持请求信号输入,高电平有效。当某一总线主控设备要占用系统总线时,通过此引脚向 CPU 提出请求。

HLDA 总线保持响应信号输出,高电平有效。这是 CPU 对 HOLD 请求的响应信号,当 CPU 收到有效的 HOLD 信号后,就会对其做出响应:一方面使 CPU 的所有三态输出的地址信号、数据信号和相应的控制信号变为高阻状态(浮动状态);同时还输出一个有效的 HLDA,表示处理器现在已放弃对总线的控制。当 CPU 检测到 HOLD 信号变低电平后,就立即使 HLDA 变低电平,同时恢复对总线的控制。

在 8086 最小模式下,由 M/\overline{IO} 、 \overline{RD} 和 \overline{WR} 的组合决定了当前数据传送的类型。其组合状态见表 3-4。

2) 最大模式下的引线

当 $\overline{MN}/\overline{MX}$ 加上低电平时,8086 工作在最大模式下。此时,引脚 24 ~ 34 的信号功能为图 3.6 中括号内所示。

QS_1 、 QS_0 指令流队列状态输出。如前所述,8086 在执行当前指令的同时,从存储器预取后边的指令放在指令流队列中。 QS_1 和 QS_0 提供指令流队列的状态信号,以便外部逻辑跟踪 CPU 内部的指令流队列。 QS_1 、 QS_0 表示的状态见表 3-5。

表 3-4 M/ $\overline{\text{IO}}$ 、 $\overline{\text{RD}}$ 和 $\overline{\text{WR}}$ 的组合决定的数据传送类型

M/ $\overline{\text{IO}}$	$\overline{\text{RD}}$	$\overline{\text{WR}}$	传送类型
0	0	1	读 I/O 端口
0	1	0	写 I/O 端口
1	0	1	读存储器
1	1	0	写存储器

表 3-5 QS_1 、 QS_0 的组合及对应的操作

QS_1	QS_0	操 作
0	0	无操作
0	1	队列中操作码的第一个字节
1	0	队列空
1	1	队列中非第一个操作码字节

$\overline{\text{S}}_2$ 、 $\overline{\text{S}}_1$ 、 $\overline{\text{S}}_0$: 总线周期状态信号输出,低电平有效,三态。这 3 个信号连接到总线控制器的输入端,译码后可以产生系统总线所需要的各种控制信号。 $\overline{\text{S}}_2$ 、 $\overline{\text{S}}_1$ 、 $\overline{\text{S}}_0$ 的代码组合以及对应的操作见表 3-6。

表 3-6 $\overline{\text{S}}_2$ 、 $\overline{\text{S}}_1$ 、 $\overline{\text{S}}_0$ 的组合及对应的操作

$\overline{\text{S}}_2$	$\overline{\text{S}}_1$	$\overline{\text{S}}_0$	对应的操作
0	0	0	发中断响应信号
0	0	1	读 I/O 端口
0	1	0	写 I/O 端口
0	1	1	暂停
1	0	0	取指令
1	0	1	读存储器
1	1	0	写存储器
1	1	1	无作用

$\overline{\text{LOCK}}$ 总线封锁信号输出,低电平有效。该信号有效时,CPU 锁定总线,不允许其他的总线控制设备申请使用系统总线。 $\overline{\text{LOCK}}$ 信号由前缀指令“LOCK”产生,并维持到 $\overline{\text{LOCK}}$ 指令后面的一条指令执行完为止。另外,CPU 在 INTR 端输入的中断请求信号也会使 $\overline{\text{LOCK}}$ 端从第一个 $\overline{\text{INTA}}$ 脉冲起到第二个 $\overline{\text{INTA}}$ 脉冲结束都保持低电平,以保证在中断响应周期之后,才允许其他总线控制设备使用总线。

$\overline{\text{RQ}}/\overline{\text{GT}}_1$ 、 $\overline{\text{RQ}}/\overline{\text{GT}}_0$ 总线请求/总线响应信号引脚。每一个引脚都具有双向功能,既是总线请求输入,也是总线响应输出;但是 $\overline{\text{RQ}}/\overline{\text{GT}}_0$ 比 $\overline{\text{RQ}}/\overline{\text{GT}}_1$ 具有更高的优先权。这些引脚内部都有上拉电阻,所以在不使用时可以悬空。这两个引脚的功能如下:

当其他的总线控制设备要使用系统总线时,就会产生一个总线请求信号(一个时钟周期宽的负脉冲),并把它送到 $\overline{\text{RQ}}/\overline{\text{GT}}$ 引脚,类似于最小模式下的 HOLD 信号。CPU 检测到总线请求信号后,在下一个 T_4 或 T_1 期间,在 $\overline{\text{RQ}}/\overline{\text{GT}}$ 引脚送出总线响应信号(一个时钟周期宽的负脉冲)给请求总线的设备,它类似于最小模式下的 HLDA 信号。然后从下一个时钟周期开

始, CPU 释放总线。总线请求设备使用完总线后, 再产生一个 $\overline{RQ}/\overline{GT}$ 信号。CPU 检测到该信号后, 从下一个时钟周期开始重新控制总线。

此外, 在最大模式下, \overline{RD} 引脚不再使用。

3. 两种工作模式下的系统总线结构

1) 最小模式下的总线结构

这种模式下的 8086 系统构成如图 3.7 所示。图中, CPU 的 20 根地址信号线通过 3 片 8282(或 74LS373)锁存器与系统的地址总线相连。16 位数据线通过一片 8286(或 74LS245)双向总线驱动器连接到系统的数据总线上。因最小模式为单处理器模式, 小系统所需的全部控制信号由 CPU 直接产生, 可直接接入总线。这样, 就构成了一个最小模式下的 8086 系统。在实际的系统中, 还应考虑以下两个问题:

① 系统总线的控制信号是 8086 CPU 直接产生的。若 8086 CPU 驱动能力不够, 可以加总线驱动器 74LS244, 以提高驱动能力。

② 按此构成的系统尚不能进行 DMA(Direct Memory Access)传送, 因为未对系统总线形成器件(8282、8286)做进一步控制。

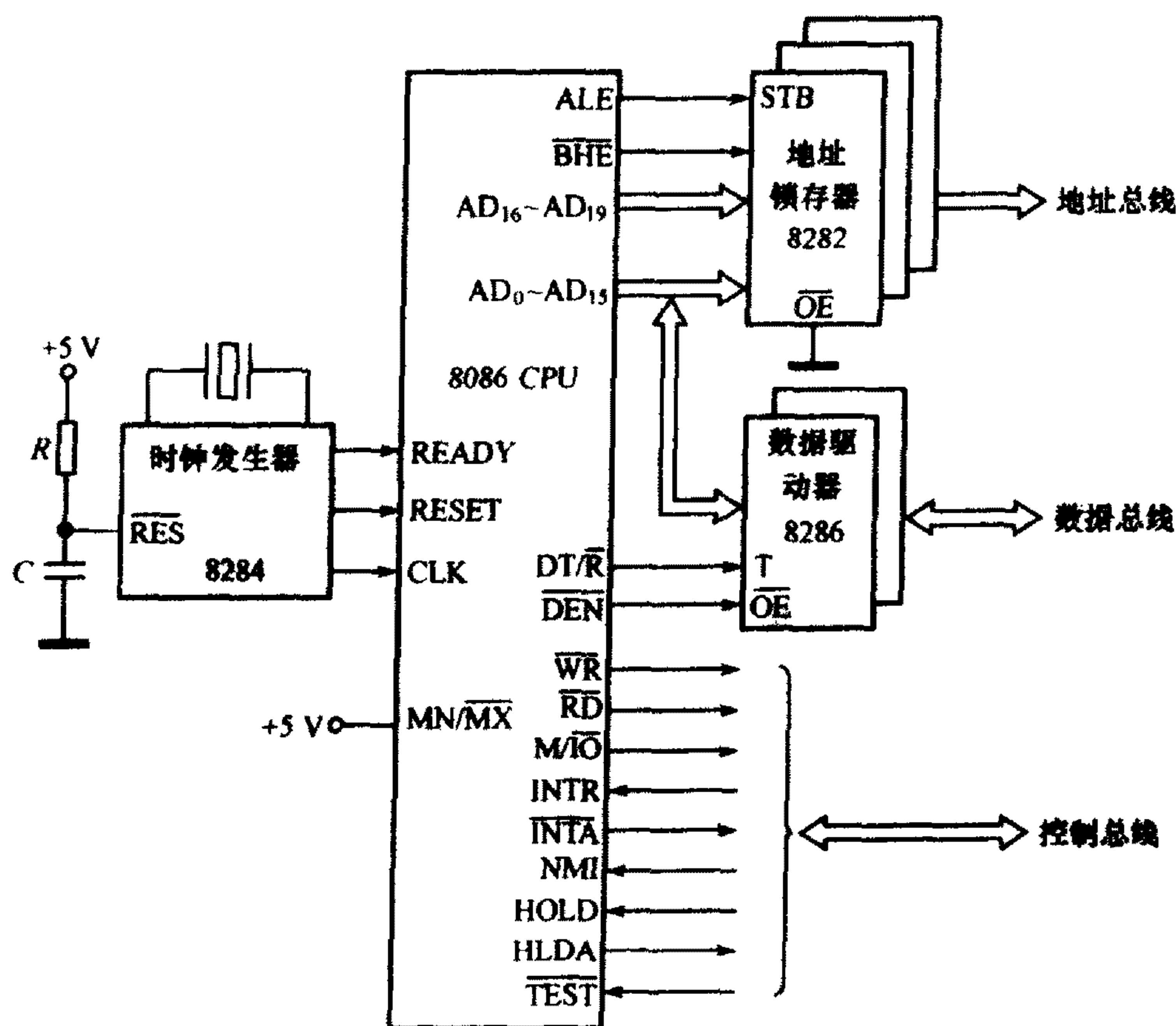


图 3.7 最小模式下的 8086 系统构成示意图

2) 最大模式下的总线结构

最大模式是支持多处理器的工作模式,即系统中可以有两个或多个能同时执行指令的处理器,增加的处理器可以是 8086 CPU,也可以是数字协处理器 8087 或 I/O 处理器 8089 等。IBM PC 系列微型计算机中的处理器就工作在最大模式,系统中配置了数字协处理器 8087,提高了系统的数据处理(特别是浮点数的处理)能力。

与单处理器系统不同的是,在多处理器系统中可能会存在多个处理器同时需要使用总线、处理器间的信息传递等问题。因此需要增加专门的控制逻辑电路来确保每次只有一个处理器占用总线,确保一个处理器能够准确地将任务分配给另一个处理器或从另一个处理器中取回结果(有关总线仲裁控制的问题将在第 4 章中介绍)。

为支持多处理器结构,当工作在最大模式下时,CPU 通过总线控制器 8288 与系统的控制总线相连,由总线控制器提供所有总线控制信号和命令信号。其系统总线结构如图 3.8 所示。图中的 8282 和 8286 也可分别用 74LS373 和 74LS245 代替。当然,这里同样没有考虑在系统总线上实现 DMA 传送的问题。在进行 DMA 传送时,需要保证总线形成电路所有输出信号都呈现高阻状态,即放弃对系统总线的控制(有关 DMA 的工作原理将在第 7 章中介绍)。

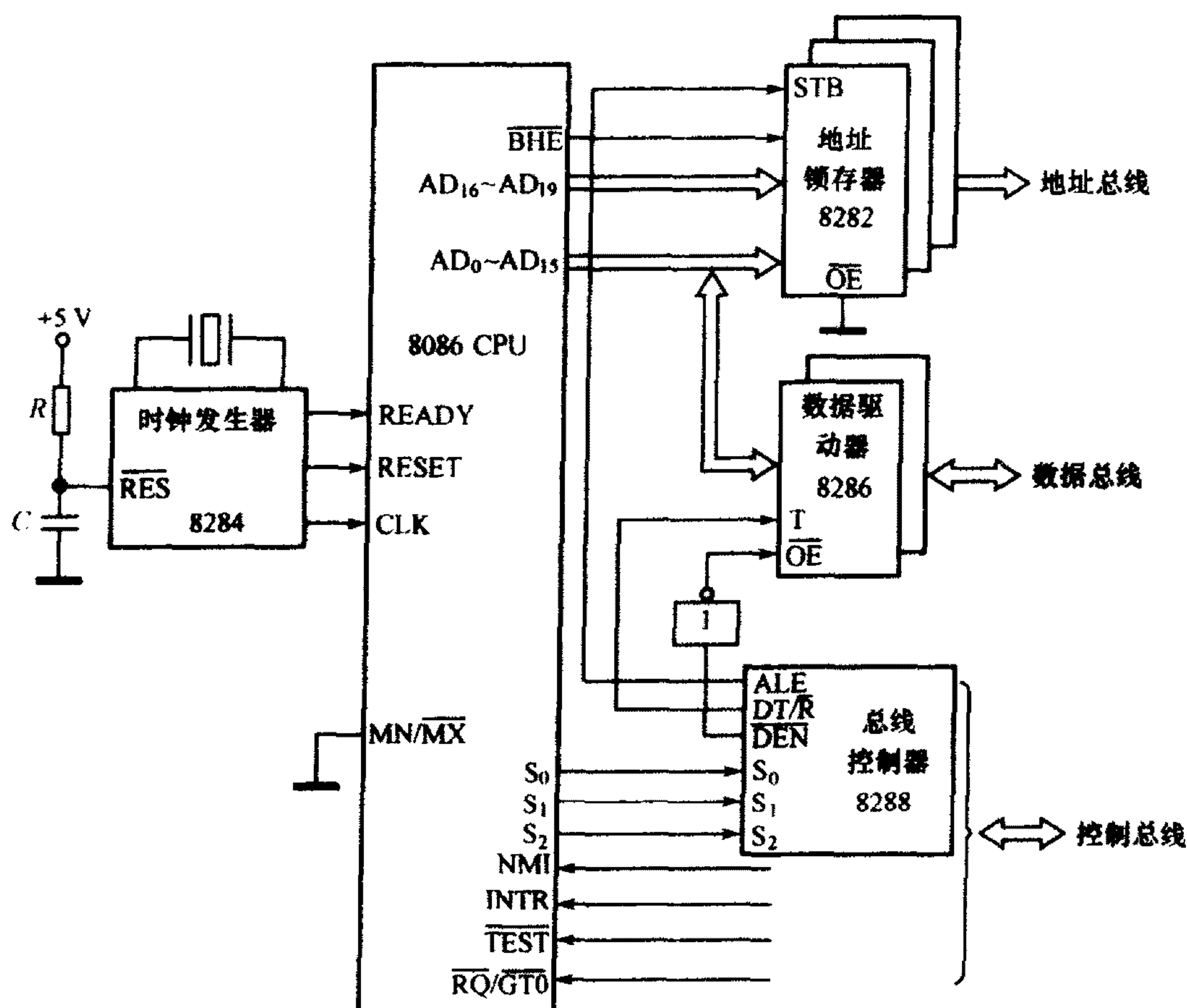


图 3.8 最大模式下的 8086 系统构成示意图

3.2.3 工作时序

微处理器是在统一的时钟信号 CLK 控制下,按照一定的时序来工作的。8086 的时钟频率为 5 MHz,故一个时钟周期等于 200 ns。CPU 的时序分为两种:时钟周期和总线周期。8086 CPU 与内存或接口间的通信都是通过总线来进行的,如将一个字节写入内存单元,或从内存某单元读一个字节到 CPU,这种通过总线对存储器或 I/O 接口进行一次访问所需的时间叫做一个总线周期,一个总线周期包括多个时钟周期。CPU 每执行一条指令至少要访问一次存储器(取指令),即至少要进行一次读存储器操作,占用一个读总线周期。

一般地,一条指令的执行需要若干个总线周期才能完成。而一个总线周期又由若干个时钟周期构成。微处理器在运行过程中是按照一个统一的时钟一步步地执行每一个操作的。每个时钟脉冲的持续时间就称为一个时钟周期。8086 CPU 的一个读(或写)总线周期至少包括 4 个时钟周期,即 T_1 、 T_2 、 T_3 和 T_4 。显然,时钟周期越短,CPU 执行的速度就越快。

典型的总线周期如图 3.9 所示。

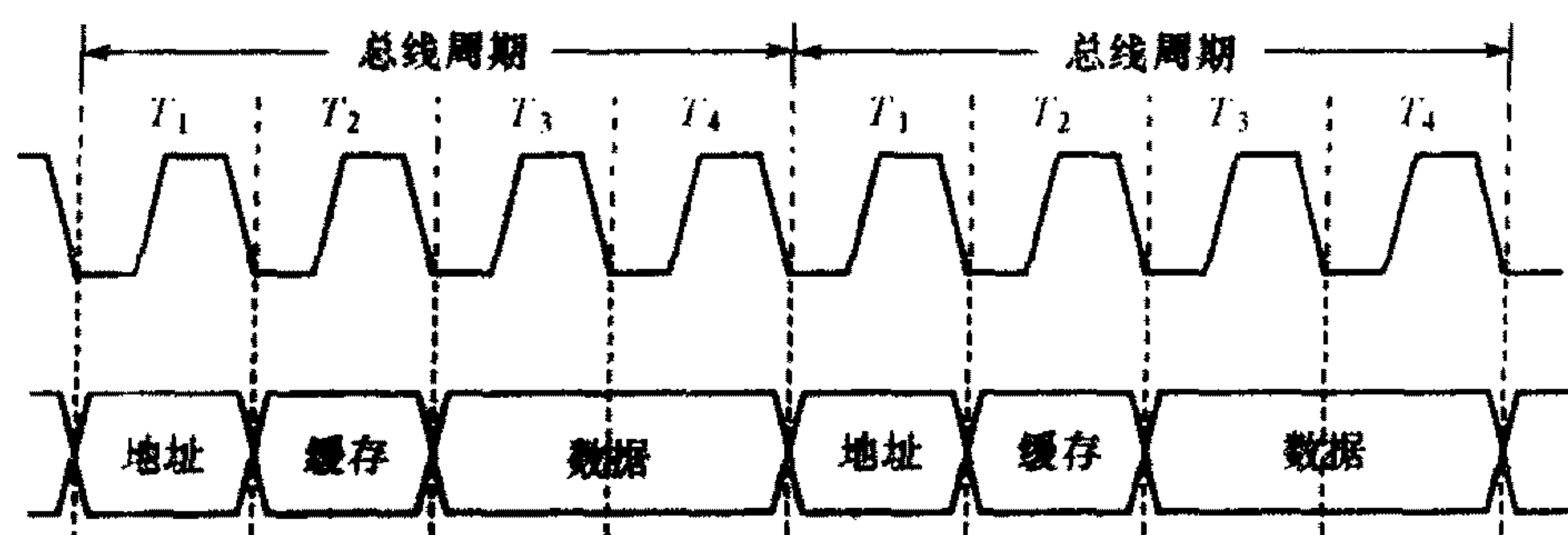


图 3.9 典型的总线周期

8086 的数据总线和部分地址总线是分时复用的。在一个总线周期内,先利用总线传送地址,将地址锁存后,再利用同一总线传送数据。即在 T_1 期间,BIU 部分将要访问的存储单元或输入/输出端口地址送上总线,若为读周期,则在第二个时钟周期将总线置为高阻缓冲状态,以使 CPU 有时间从输出地址方式转换为输入数据方式。之后在 T_3 到 T_4 期间从总线读入数据到 CPU;若为写周期,则 CPU 就不必转换工作方式,在地址锁存后直接输出数据到总线上。

只有在指令流队列出现 2 个以上空单元时要填补指令流队列或在执行指令的过程中需要申请一个总线周期时,BIU 部分才会进入执行总线周期的状态。在两个总线周期之间,有时可能会出现一些总线上没有信息传送的时钟周期,此时的总线状态称为空闲状态。

1. 最小模式下的工作时序

8086 CPU 在最小模式下的读总线周期和写总线周期时序图分别如图 3.10 和图 3.11 所示。

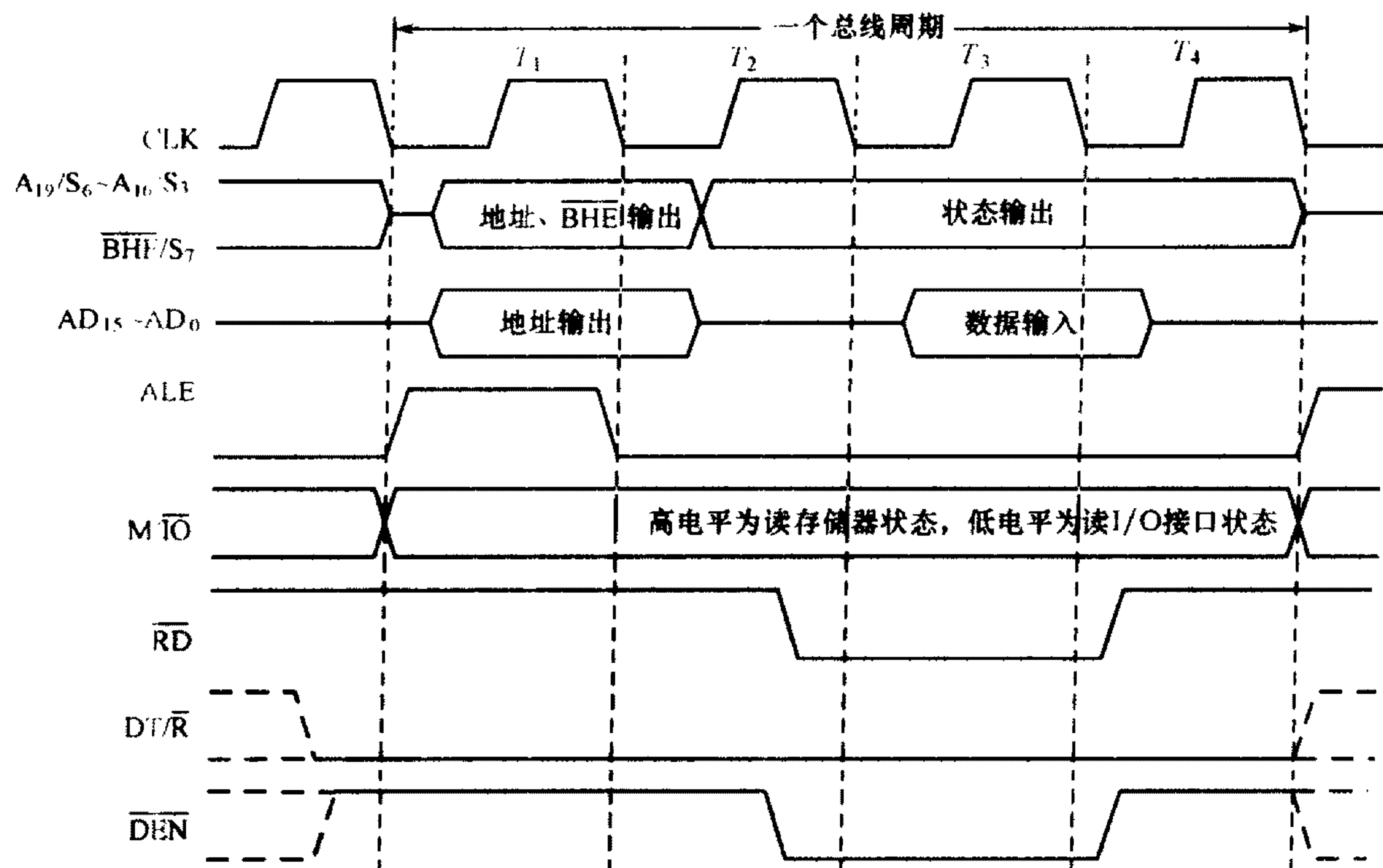


图 3.10 8086 最小模式下的读总线周期

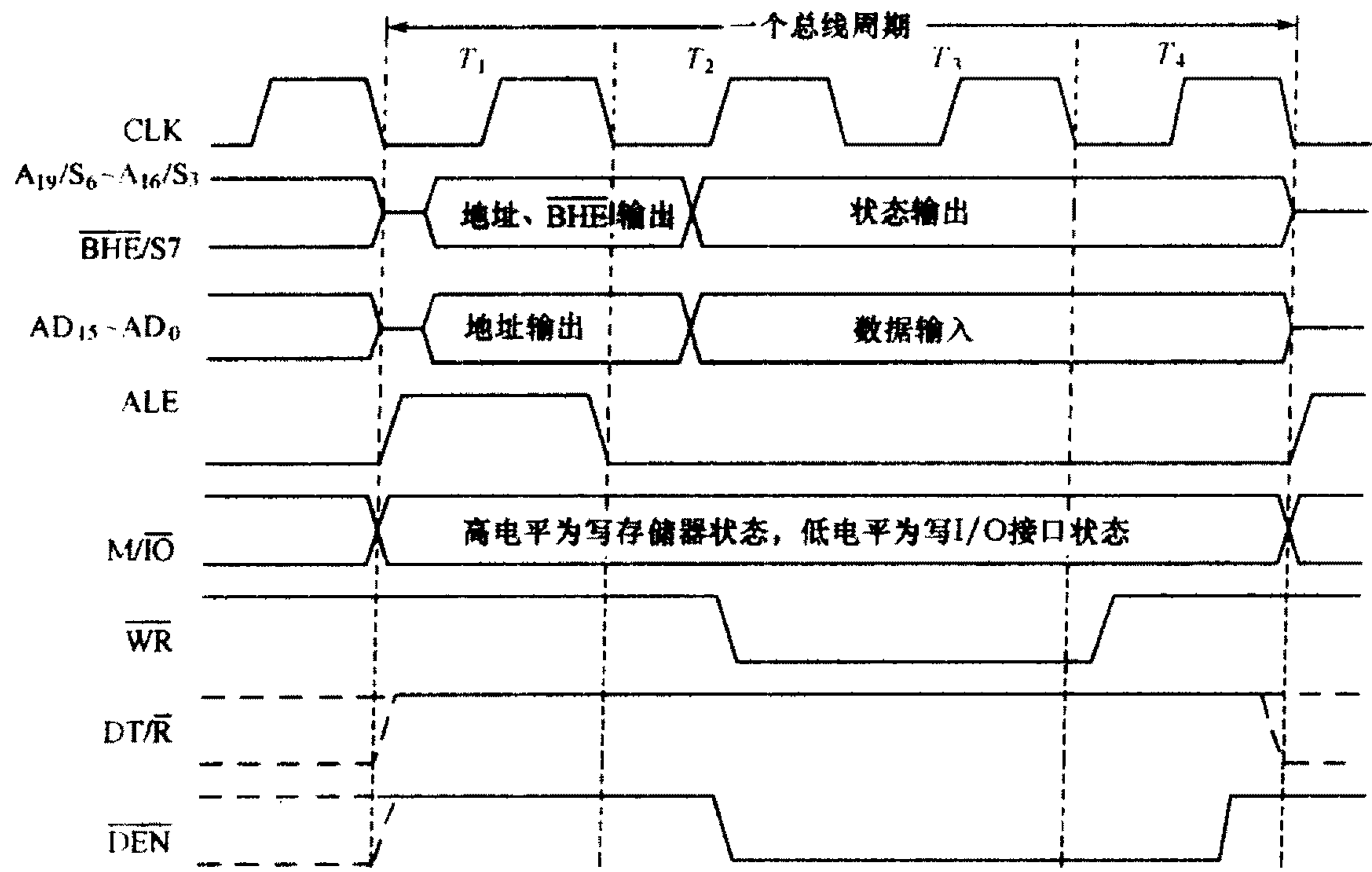


图 3.11 8086 最小模式下的写总线周期

由前述可知,一个正常的 8086 读(写)总线周期至少需要 4 个时钟周期($T_1 \sim T_4$)。在 T_1 期间,地址/状态复用信号线 $A_{19}/S_6 \sim A_{16}/S_3$ 和地址/数据复用信号线 $AD_{15} \sim AD_0$ 分别送出地址 $A_{19} \sim A_{16}$ 和 $A_{15} \sim A_0$,与此同时送出地址锁存允许信号 ALE。BHE 的状态由 BHE/ S_7 端输出。外部电路在 T_1 后期利用 ALE 的下降沿把地址信号锁存到地址锁存器中,从而在锁存器的输出端得到完整的 20 位地址信号 $A_0 \sim A_{19}$ 。一旦 ALE 的高电平将地址信号锁存,在此后的时钟周期里,即可利用有关的控制信号如 $\overline{M}/\overline{IO}$ 、 \overline{RD} 、 \overline{WR} 等完成对内存或外设的读/写操作。读总线周期中,CPU 在 T_3 到 T_4 期间读入总线上的数据。在写总线周期中,CPU 从 T_2 开始把数据送到总线上并维持到 T_4 。

某些情况下,当内存或接口的速度比较慢,使得在 4 个时钟周期里不能完成读/写操作时,可通过时钟发生器(8284)产生一个低电平信号送到 8086 的 READY 端。8086 在每个总线周期的 T_3 开始处都要检查 READY 的状态。若此时 READY 为低电平,则 CPU 不执行 T_4 ,而是在 T_3 之后插入一个等待时钟周期 T_w ,以等待存储器或 I/O 接口完成读/写操作。在 T_w 的开始时刻,CPU 还要检查 READY 状态,若仍为低电平,则再插入一个 T_w 。此过程一直进行到某个 T_w 开始时,READY 已经变为高电平,这时下一个时钟周期就是总线周期的最后一个时钟周期 T_4 (如图 3.12 所示)。由此可见,利用 READY 信号,CPU 可以插入若干个 T_w ,使总线周期延长,达到可靠地读/写内存和 I/O 接口的目的。

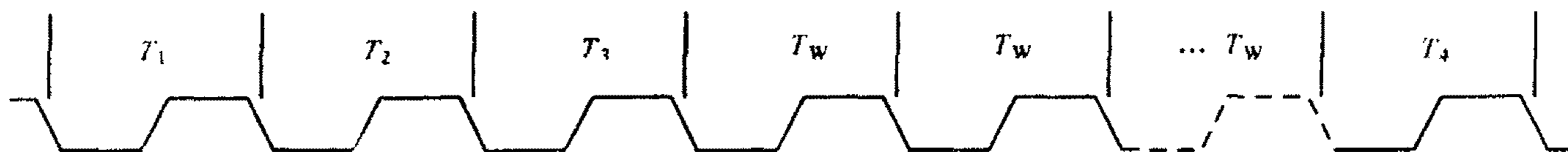


图 3.12 插入 T_w 的总线周期

另外还要注意一点,CPU 的读(\overline{RD})或写(\overline{WR}),是在 T_4 开始时刻(或 \overline{RD} 、 \overline{WR} 信号的后沿)进行的,这时数据线上的数据已经到达稳定状态,只有这样,利用 READY 插入 T_w 周期才有意义。

2. 最大模式下的工作时序

8086 在最大模式下的工作时序与最小模式下的工作时序有以下几个不同点:

① 所有地址锁存器和数据驱动器的控制信号在最大模式下都由总线控制器 8288 根据 CPU 输出的 3 个状态位 S_0 、 S_1 、 S_2 的状态产生的;

② 最小模式下的 $\overline{M}/\overline{IO}$ 、 \overline{RD} 和 \overline{WR} 信号将分别由总线控制器的存储器读命令 \overline{MRDC} 和 I/O 读命令 \overline{IORC} 、存储器写命令 \overline{MWTC} 和先行存储器写命令 \overline{AMWC} 及 I/O 写命令 \overline{IOWC} 、先行 I/O 写命令 \overline{AIOWC} 代替;

③ 总线控制器输出的数据允许信号 $\overline{\text{DEN}}$ 与最小模式下 CPU 产生的 $\overline{\text{DEN}}$ 信号极性相反。
8086 在最大模式下的读总线周期时序图分别如图 3.13 和图 3.14 所示。

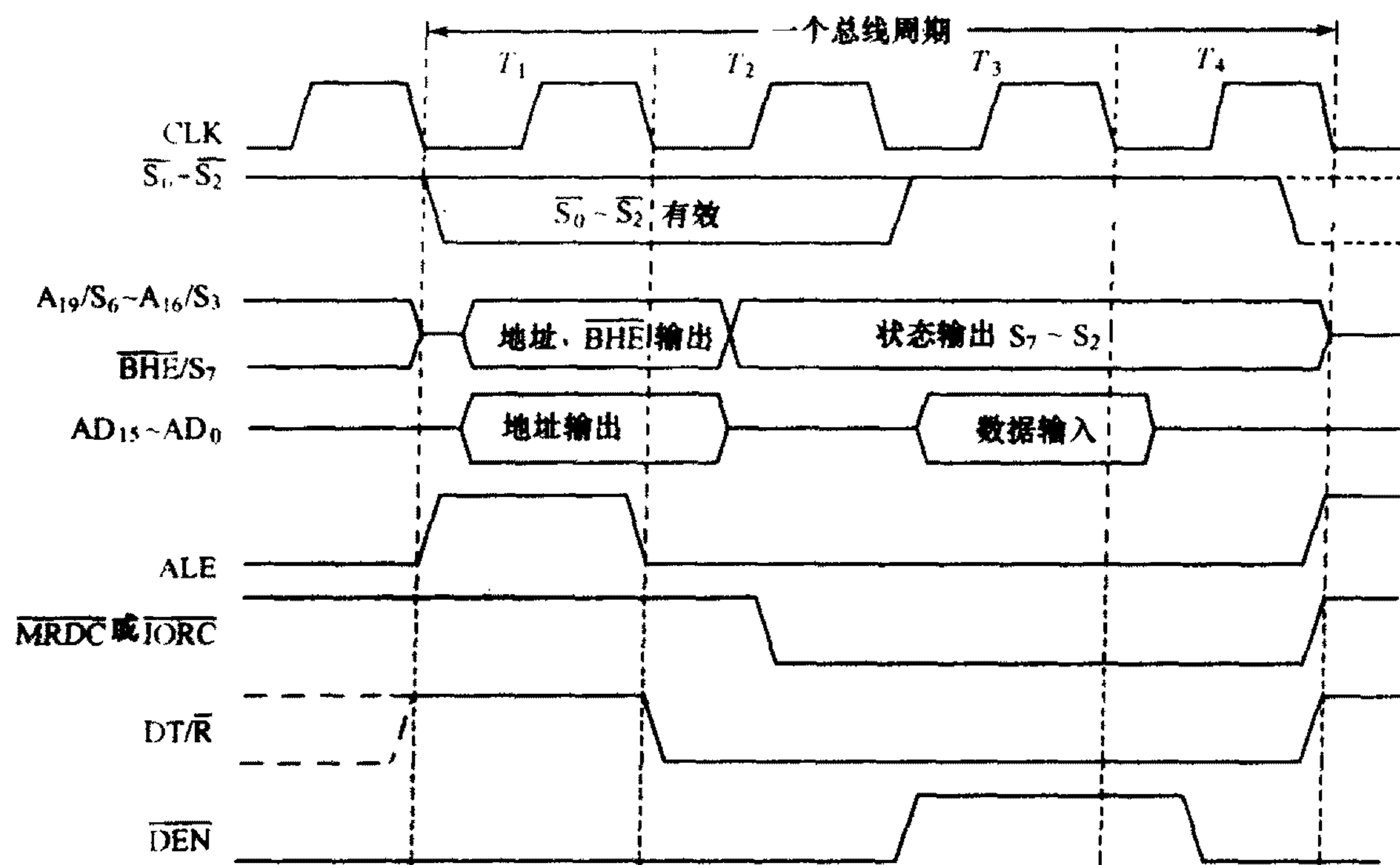


图 3.13 8086 最大模式下的读总线周期

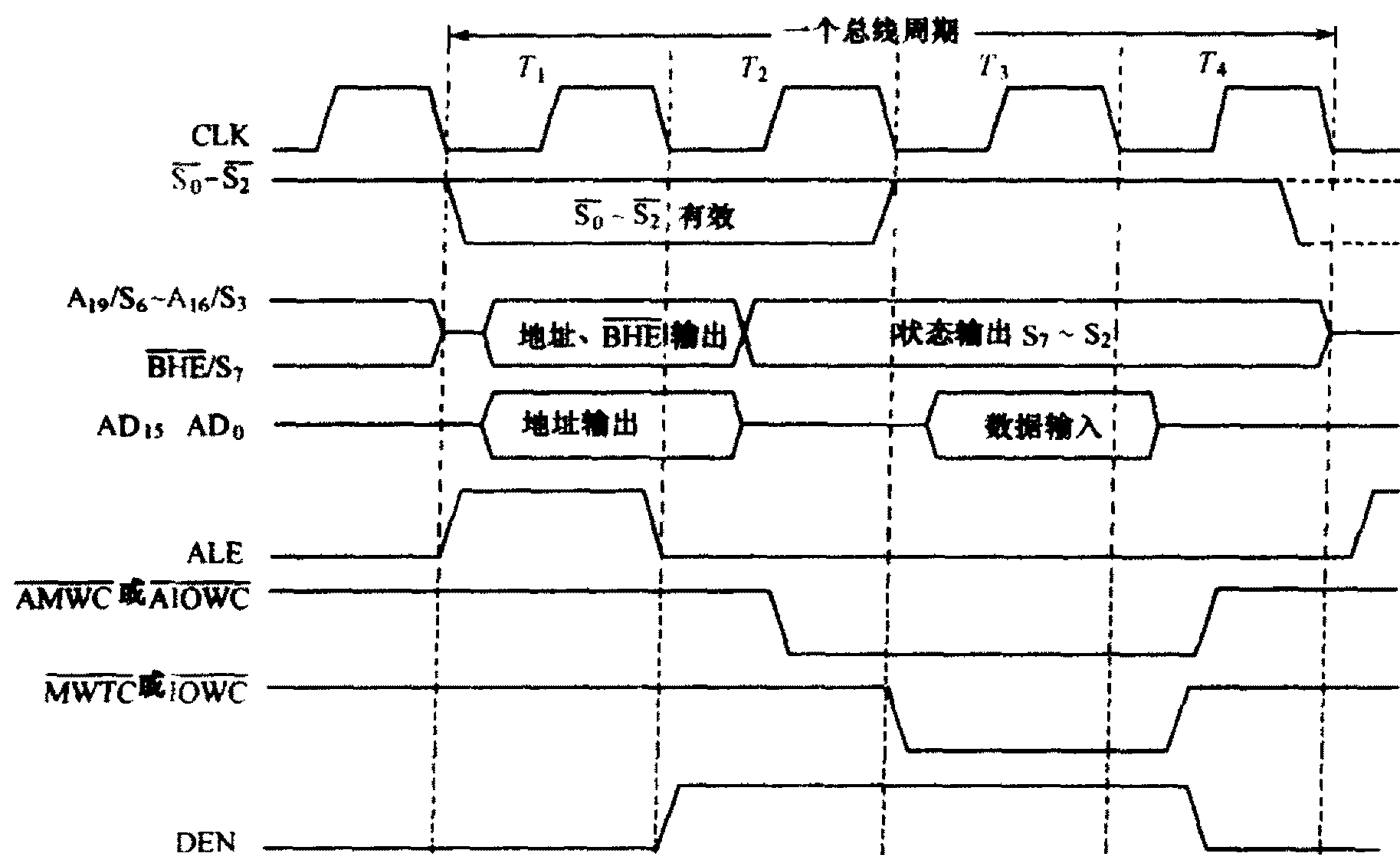


图 3.14 8086 最大模式下的写总线周期

3.3 8086 的寄存器组

8086 内部共有 14 个 16 位寄存器。按其功能可分为三大类,即通用寄存器组、段寄存器组以及控制寄存器组,其结构如图 3.15 所示。

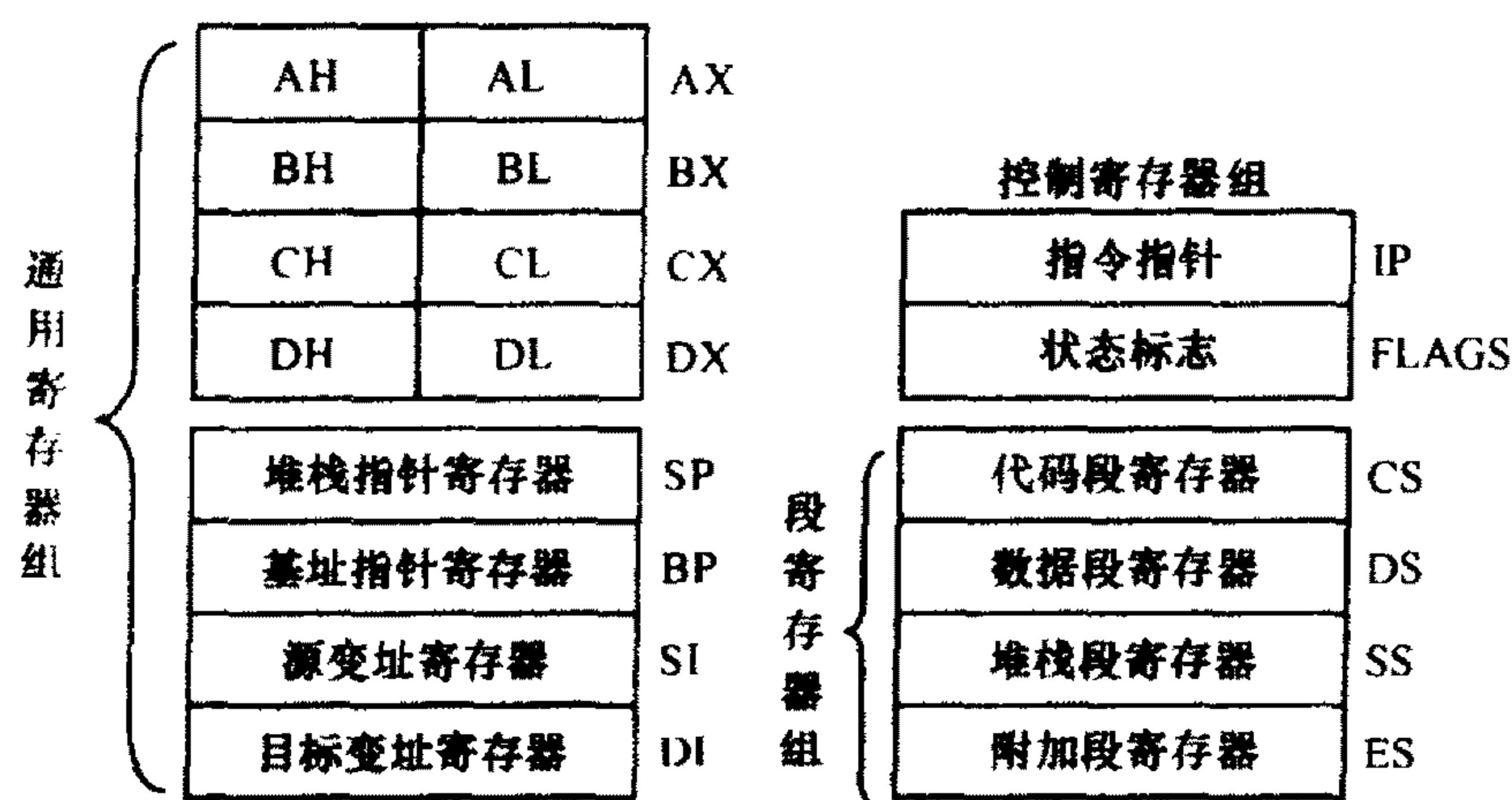


图 3.15 8086 内部寄存器结构

3.3.1 通用寄存器

通用寄存器组包括 4 个数据寄存器、2 个地址指针寄存器和 2 个变址寄存器。

1. 数据寄存器 AX、BX、CX、DX

数据寄存器一般用于存放参与运算的数据或运算的结果。每一个数据寄存器都是 16 位寄存器,但又可将高 8 位和低 8 位分别作为两个独立的 8 位寄存器使用。它们的高 8 位记为 AH、BH、CH、DH,低 8 位记为 AL、BL、CL、DL。这种灵活的使用方法给编程带来极大的方便,既可以处理 16 位数据,也能处理 8 位数据。

数据寄存器除了作为通用寄存器使用外,它们还有各自特定的用法:

① AX(Accumulator)称为累加器,常用于存放算术逻辑运算中的操作数。所有的 I/O 指令都使用累加器与外设接口传送信息。

② BX(Base)称为基址寄存器,常用来存放访问内存时的基地址。

③ CX(Count)称为计数寄存器,在循环和串操作指令中用做计数器。

④ DX(Data)称为数据寄存器,在寄存器间接寻址的 I/O 指令中存放 I/O 端口的地址。

另外,在做双字长乘、除法运算时,DX 与 AX 合起来存放一个双字长数(32 位),其中 DX

存放高 16 位,AX 存放低 16 位。

2. 地址指针寄存器 SP、BP

SP(Stack Pointer)称为堆栈指针寄存器,它在堆栈操作中用来存放栈顶的偏移地址,永远指向堆栈的栈顶。

BP(Base Pointer)称为基地址指针寄存器。一般也常用来存放访问内存时的基地址。但它通常是与 SS 寄存器配对使用(BX 通常是与 DS 寄存器配对使用)。

作为通用寄存器,SP 和 BP 也可以存放数据。但实际上,它们更重要的用途是存放内存单元的偏移地址,特别是在访问堆栈时作为指向堆栈的指针(因为它们默认的段寄存器都是堆栈段寄存器 SS)。

3. 变址寄存器 SI、DI

SI(Source Index)称为源变址寄存器,DI(Destination Index)称为目标变址寄存器,它们常常在变址寻址方式中作为索引指针。在字符串操作指令中,要求用 SI 作为源变址寄存器,存放源操作数的偏移地址;DI 作为目标变址寄存器,存放目标操作数的偏移地址。

以上 8 个 16 位寄存器作为通用寄存器都具有相同的功能,即存放操作数及运算结果,除此之外,在不同的场合它们各自又有自己独特的用法。

3.3.2 段寄存器组

在 3.2.1 节中曾提到,存储器是分段管理的。每一个存储单元的地址都是由它所在逻辑段的段基地址和段内的偏移地址两部分构成。CPU 在访问存储器时,首先要找到访问单元所在的段,也就是确定单元所在的段基地址。段寄存器就是用来存放段基地址的,即逻辑段起始地址的高 16 位。8086 共有 4 个 16 位段寄存器,分别是代码段寄存器 CS(Code Segment)、数据段寄存器 DS(Data Segment)、附加段寄存器 ES(Extra Segment)和堆栈段寄存器 SS(Stack Segment)。

① 代码段寄存器 CS 代码段存放的是当前执行程序的指令代码。CS 的内容是代码段的段基地址,它和指令指针 IP 一起决定下一条所要执行指令的物理存储地址。

② 数据段寄存器 DS 数据段通常用来存放数据和字符。DS 存放当前数据段的段基地址。

③ 附加段寄存器 ES 附加段是一个附加数据段,主要用在字符串操作时作为目标地址使用。ES 的内容就是加段的段基地址。

④ 堆栈段寄存器 SS 堆栈是在存储器中开辟的一个特殊存储区,用于存放当前暂时不用但又需要保存的数据和地址。如在子程序调用或响应中断时需要保存返回主程序的地址和进入子程序后将要改变其值的寄存器的内容。堆栈的操作遵循先进后出的原则,操作的地址由 SS(堆栈段基地址寄存器)和 SP(堆栈指针寄存器)的内容指定。

3.3.3 控制寄存器

8086 内部包括指令指针 IP 和标志寄存器 FLAGS 两个控制寄存器。

1. 指令指针 IP (Instruction Pointer)

IP 用来存放下一条要执行指令的偏移地址。CPU 取指令时总是以 CS 的内容为段基地址,以 IP 为段内偏移地址。当 CPU 从 CS 段偏移地址为(IP)的内存单元中取出指令代码的一个字节后,IP 自动加 1,指向指令代码的下一个字节。如果遇到过程调用、转移及返回等指令时,系统将根据程序确定新的 IP 的内容,使其不再加 1。

用户程序不能直接访问 IP,即指令中的操作数不能是 IP。

2. 标志寄存器 FLAGS

FLAGS 称为标志寄存器或程序状态字(PSW),是一个 16 位寄存器,但只使用了其中的 9 位,包括 6 个状态标志位和 3 个控制标志位,如图 3.16 所示。

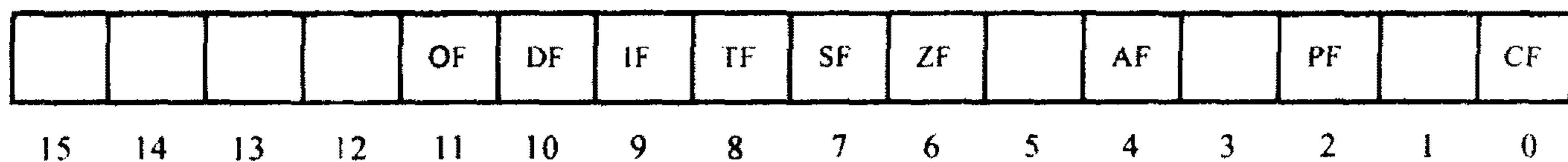


图 3.16 8088/8086 的标志寄存器

1) 状态标志位

状态标志位记录了算术和逻辑运算结果的一些特征。例如,结果是否为 0、是否有进位或借位、结果是否溢出等。不同的指令对状态标志位具有不同的影响。

CF (Carry Flag) 进位标志位。在进行加(减)法运算时,若最高位向更高位有进(借)位时 $CF = 1$,否则 $CF = 0$ 。

PF (Parity Flag) 奇偶标志位。当运算结果的低 8 位中“1”的个数为偶数时 $PF = 1$,为奇数时 $PF = 0$ 。

AF (Auxiliary Carry Flag) 辅助进位标志位。在加(减)法操作中,若 b_3 向 b_4 有进位(借位)时 $AF = 1$,否则 $AF = 0$ 。用 DAA 和 DAS 指令可测试这个标志位,以便在 BCD 加法或减法之后调整 AL 中的值。

ZF (Zero Flag) 零标志位。当运算结果各位均为零时 $ZF = 1$,否则 $ZF = 0$ 。

SF (Sign Flag) 符号标志位。当运算结果的最高位(字节操作时是 b_7 ,字操作时是 b_{15})为 1 时 $SF = 1$,否则 $SF = 0$ 。即它和运算结果的最高位相同。因为在补码运算时最高位为符号位, $SF = 1$ 表示结果为负; $SF = 0$ 表示结果为正。

OF(Overflow Flag) 溢出标志位。当算术运算的结果超出了带符号数的范围,即溢出时 $OF = 1$, 否则 $OF = 0$ 。8 位带符号数的范围是 $-128 \sim +127$, 16 位带符号数的范围是 $-32768 \sim +32767$ 。

下面以一个简单的算术运算例来说明操作结果对状态标志位的影响。

例 3-1 假设执行一条加法指令, 计算 $5439H + 476AH$ 后各状态标志位的状态为何?
解

$$\begin{array}{r}
 0101 \quad 0100 \quad 0011 \quad 1001 \\
 + 0100 \quad 0111 \quad 0110 \quad 1010 \\
 \hline
 1001 \quad 1011 \quad 1010 \quad 0011
 \end{array}$$

则执行这条加法指令后标志寄存器的状态为: $CF = 0, PF = 1, AF = 1, ZF = 0, SF = 1, OF = 1$ 。

2) 控制标志位

控制标志位用于设置控制条件。控制标志被设置后便对其后的操作产生控制作用。

TF(Trap Flag) 陷阱标志位, 也称为跟踪标志位。TF = 1 时, CPU 处于单步执行指令的工作方式。这种方式便于进行程序的调试, 每执行一条指令, 自动产生一次单步中断, 从而使用户能逐条地检查指令。

IF(Interrupt Enable Flag) 中断允许标志位。IF = 1 时, CPU 可以响应可屏蔽中断请求; IF = 0 时, 则 CPU 禁止响应可屏蔽中断请求。IF 的状态对不可屏蔽中断及内部中断没有影响。

DF(Direction Flag) 方向标志位。DF = 1 时, 串操作按减地址方式进行, 也就是说, 从高地址开始, 每处理一个元素, 地址指针就自动减 1 (或减 2); DF = 0 时, 串操作按增地址方式进行。

3.4 存储器组织

8086 可寻址 1 MB 的存储器空间。要做到这一点, 最基本的是每个空间要有惟一的地址, 即地址编码至少要是 20 位二进制数 ($2^{20} = 1 \text{ MB}$)。本节介绍 20 位地址的形成及存储器中的分段组织。

3.4.1 物理地址与存储器的分段

8086 的存储器是以字节为单位组织的, 称为“字节寻址”的计算机。每个单元存放一个字节数, 即 8 位二进制编码, 并具有一个惟一确定的地址, 称为物理地址, 为 20 位二进制 (或 5 位十六进制) 编码, 地址范围为: $00000H \sim FFFFFH$ 。

每一个单字节数在存储器中都有一个确定的字节地址。对于一个双字节(16位)数,其在存储器中的存放规律是:低字节放在低地址,高字节放在高地址。而用低字节地址表示整个双字节数的地址。如图3.17所示,16位数7788H,低字节88H放在低地址6000H:0004H,高字节放在高地址6000H:0005H中。

8086的内部寄存器、内部总线及算术逻辑单元ALU都是16位的,要实现对1MB存储空间的访问,很明显不采取特殊措施是不行的。8086采用了将地址空间分段的方法来解决这个问题,即将1MB的地址空间分为若干个逻辑段,然后用段基地址和段内偏移地址来形成确定的物理存储器。因段寄存器和CPU内部寄存器都是16位的,所以段基地址和段内偏移地址也都是16位的。

所谓**段基地址**(简称段地址或段基地址),表示的是某个逻辑段在整个存储空间中所处的位置,是该段起始地址的高16位。段的起始地址也称为**段首地址**,它必须是能够被16整除的数,亦即它的低4位二进制数必须是0。所以,将段基地址左移4位得到的就是该段的段首地址。

偏移地址(或称有效地址EA)是指段内某个单元相对于段首的距离,如图3.17所示中,存放数据88H的单元的段基地址是6000H,段首地址即为60000H,该单元与段首的距离是4个字节,故其偏移地址为0004H。

段基地址和段内偏移地址又称为**逻辑地址**,逻辑地址通常写成xxxxH:yyyyH的形式,其中xxxxH是段基地址,yyyyH是段内偏移地址(也称为偏移量),它们都是16位的无符号二进制数。20位的物理地址与逻辑地址的关系如下:

$$\text{物理地址} = \text{段基地址} \times 16 + \text{段内偏移地址}$$

段基地址乘以16相当于段基地址左移4位(或段基地址后面加4个0),然后再与偏移地址相加,即可得到20位的物理地址。其形成过程如图3.18所示。

例如:若逻辑地址为3A00H:12FBH,对应的物理地址是3B2FBH;若逻辑地址是8000H:1200H,则对应的物理地址为81200H。

值得注意的是,一个物理地址所对应的逻辑地址是不惟一的。

例如:上例中的物理地址3B2FBH的逻辑地址可以是

3A00H:12FBH

3000H:B2FBH

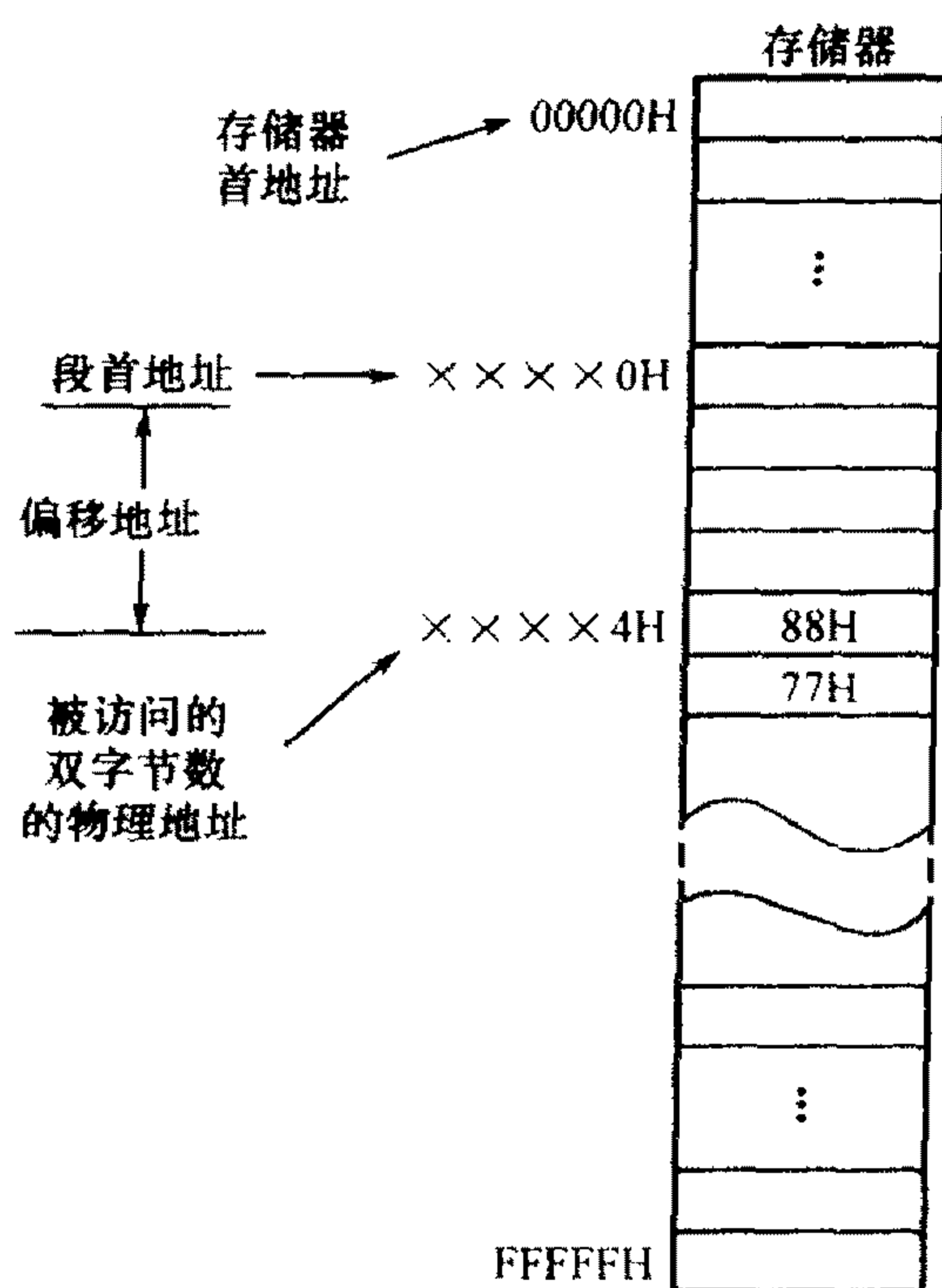


图3.17 存储器地址

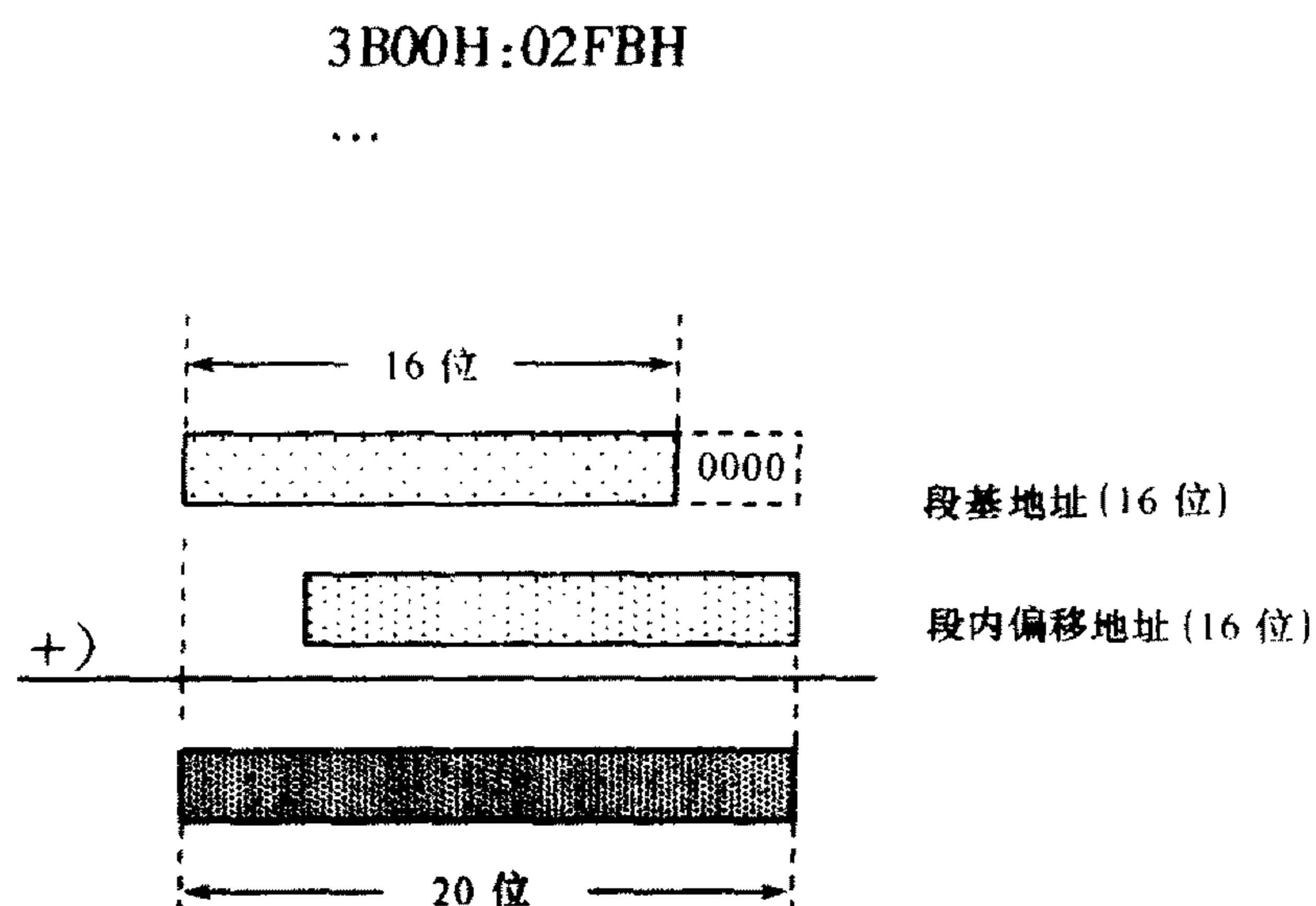


图 3.18 20 位物理地址的形成

既然一个段的起始地址的高 16 位就是该段的段地址,那么在整个 1 MB 的存储空间中就可以有 $2^{16} = 64$ K 个段地址,任意两个段地址之间的最小间隔为 16 B。因段首地址的最低 4 位为 0,且偏移地址是 16 位的,故一个逻辑段最大可为 64 KB,最小为 16 B,而且段与段之间是可以相互重叠的。如图 3.19 所示。

段地址是由段寄存器提供的。8086 有 4 个段寄存器,所以它同时可以访问 4 个存储段。通过程序改变段寄存器的内容便可实现对所有段的访问,也可使段与段之间重叠、紧密连接或间隔分开。

分段(段加偏移)寻址所带来的好处是允许程序在存储器内重定位(浮动),允许实模式下编写的程序在保护模式下运行。可重定位程序是一个不加修改就可以在任何存储区域中运行的程序。这是因为段内偏移总是相对段起始地址(段基地址)的,所以只要在程序中不使用绝对地址访问存储器,就可以把整个程序作为一个整体移到一个新的区域。在 DOS 中,程序载入到内存时由操作系统来指定段寄存器的内容,从而实现程序的重定位。

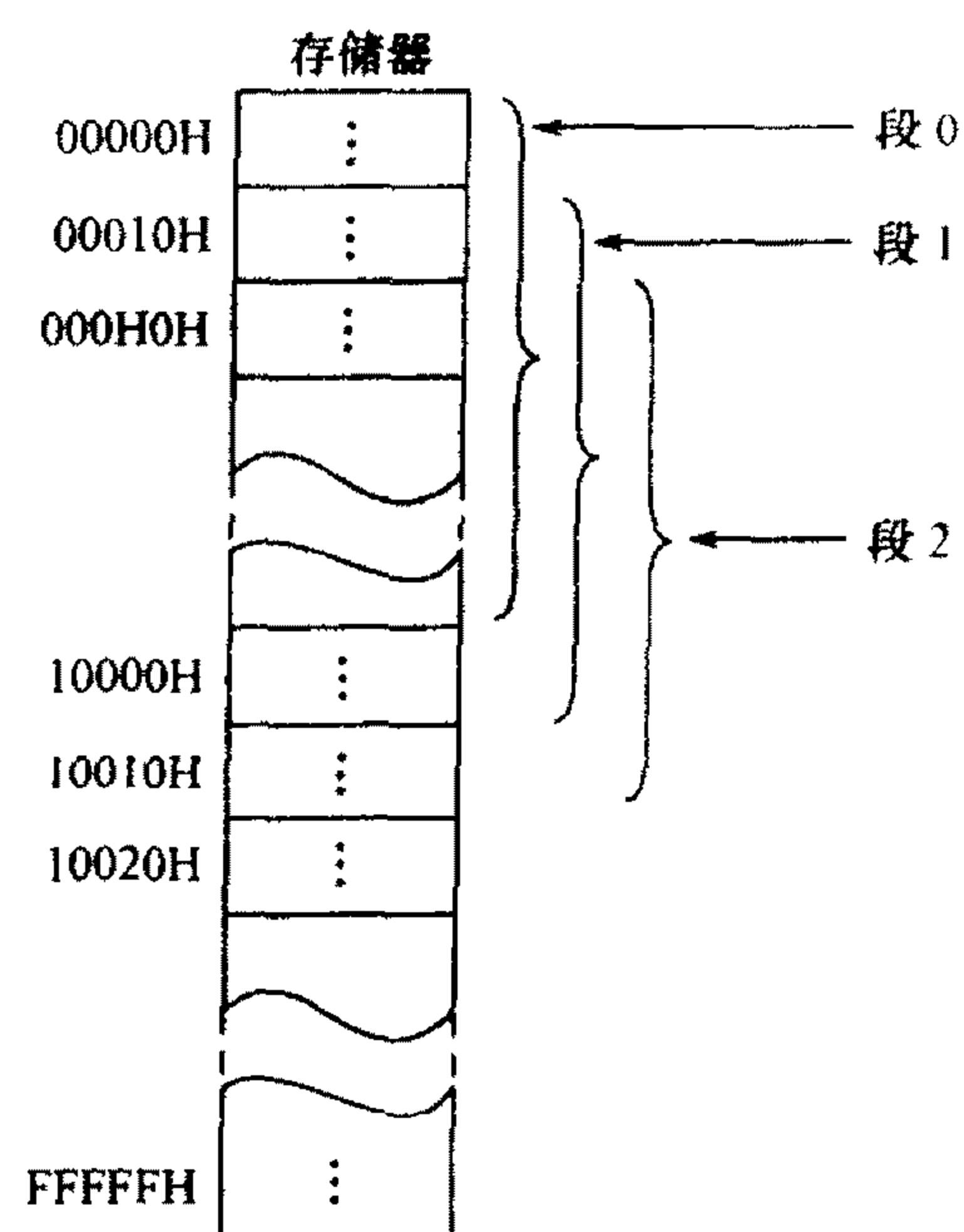


图 3.19 存储器段的覆盖示意图

3.4.2 段寄存器的使用

段寄存器的设立不仅使 8086 的存储空间扩大到 1 MB,而且为信息按特征分段存储带

来了方便。在存储器中,信息按特征可分为程序代码、数据、堆等。为了操作方便,存储器可相应地划分为:程序段——用来存放程序的指令代码;数据段——用来存放数据和运算结果;堆栈段——用来传递参数、保存数据和状态信息。有时一种类型的段可能会有多个。通过修改段寄存器的内容,可将这些段设置在存储器的任何位置上。这些段可以通过段寄存器的设置使之相互独立,也可将它们部分或完全重叠。

8086 对访问不同内存段所使用的段寄存器和相应的偏移地址的来源有一些具体约定,见表 3-7。

表 3-7 表明,访问存储器时,其段地址可以由“默认”的段寄存器提供,也可以由“指定”的段寄存器提供。当指令中没有显式地“指定”使用某一个段寄存器时,就由“默认”段寄存器来提供访问内存的段地址。在实际进行程序设计时,大多数情况都使用默认段寄存器来寻址内存。在序号为 3、5、6 这三种访问存储器的操作中,允许在指令中指定使用另外的段寄存器,这样可很灵活地访问不同的内存段。这种指定通常是靠在指令码中增加一个字节的前缀来实现。1、2、4 这三种类型的内存访问只能用默认的段寄存器。即取指令一定要使用 CS;堆栈操作一定要使用 SS;串操作指令的源和目的段基地址一定要用 DS 和 ES。

表 3-7 8086 对段寄存器使用的约定

序号	内存访问类型	默认段寄存器	可指定段寄存器	段内偏移地址来源
1	取指令	CS	无	IP
2	堆栈操作	SS	无	SP
3	源串	DS	CS、ES、SS	SI
4	目的串	ES	无	DI
5	BP 用作基地址寻址	SS	CS、ES、DS	按寻址方式计算得到的有效地址
6	一般数据存取	DS	CS、ES、SS	按寻址方式计算得到的有效地址

DS、ES 和 SS 要用传送指令来进行设置,但在用户程序中不允许设置 CS,CS 一般由操作系统进行设置。宏汇编语言中的伪指令 ASSUME 及 JMP、CALL、RET、INT 和 IRET 等指令可以改变和影响 CS 的内容。更改段寄存器的内容意味着内存段的移动,这说明无论程序段、数据段还是堆栈段,都可以不限制在 64 KB 的容量内,都可通过采用重新设置段寄存器内容的方法来扩大段,而且各内存段都可以在整个存储空间中浮动。

表中前四类内存操作的偏移地址只能使用一个 16 位的指针寄存器或变址寄存器。例如,取指令时为指令指针寄存器 IP;堆栈操作时为堆栈指针 SP;串操作时分别为 SI 和 DI。后两类内存操作,则根据不同寻址方式来计算出段内偏移。

在实际的程序设计中,每个程序模块只可以有一个代码段、一个数据段、一个堆栈段和

一个附加段。它们的最大容量分别为 64 KB。如果程序的容量小,可使各段部分重叠,例如将数据段和附加段重叠或完全重合。

3.5 80X86 微处理器

从 Intel 公司在上世纪 70 年代推出第一款实用的 8080 微处理器以来,CPU 技术的发展可谓日新月异。在 8086 进入市场后很短的时间内,它的运行速度、功能及寻址 1 MB 内存的能力等各方面就已不能满足需要。因此从 80 年代起,Intel 公司就又相继推出了 80286、80386、80486 及 Pentium 系列微处理器。本节介绍 Intel 公司在 8086 之后的一些代表产品——快速 16 位微处理器 80286、32 位微处理器 80386 以及目前最新型 CPU 的代表 Pentium 4 的主要特性及内部结构。因 80286 事实上与 8086 属于同一代产品,它所引进的新技术在其后的产品中都完全体现,所以下面将讨论的重点放在第一代 32 位微处理器 80386 以及 Pentium 4 上,对 80286 仅做一般性介绍。

3.5.1 80286 微处理器

1. 主要性能

80286 是比 8086/8088 更为先进的 16 位微处理器。芯片上集成了 13.5 万只晶体管。对外具有 68 根引脚,为 4 列直插式封装,时钟频率 8 MHz ~ 10 MHz。与 8086 相比,80286 CPU 主要具有如下几个特点:

- ① 80286 CPU 有 24 位地址线、16 位数据线,且地址与数据线不再复用。
- ② 对 8086 向上兼容。具有 8086/8088 CPU 的全部功能,在 8086/8088 上运行的汇编语言程序不需修改就可在 80286 CPU 上运行。
- ③ 可运行在实地址和保护虚地址两种模式下。80286 CPU 片内具有存储器管理部件 (Memory Management Unit, MMU) 和保护结构。能够实现在实地址和保护虚地址两种模式下访问存储器。在实地址模式下,80286 相当于一个快速的 8086,可寻址 1 MB 的物理地址空间,同样采用分段的方法管理存储器,每个段最大为 64 KB,从逻辑地址到物理地址的转换也与 8086 一样。

当运行在保护虚地址模式时,80286 可直接寻址的存储器空间为 16 MB,并可提供 1 GB (2^{30} 字节) 的虚拟地址空间。其保护功能包括段寄存器的保护;任务之间、任务与操作系统之间、任务内程序与数据之间的分离与保护;对存储器的访问保护及特权级保护等。所以,80286 CPU 能够可靠地支持多用户、多任务系统。

④ 首次具备虚拟存储器管理功能。在 8086 中,程序占有的存储器和 CPU 可访问的存储器是一致的。从 80286 开始,CPU 能够支持对虚拟存储器的访问。所谓虚拟存储器是一种设计技术,它通过 CPU 内部的存储器管理部件 MMU 向用户程序提供比实际内存储器大得多的存储器空间。虚拟存储器管理要解决的问题是将较小的物理存储器空间分配给具有较大虚拟存储器空间的多用户、多任务(有关虚拟存储器管理的更进一步的概念将在第 6 章中介绍)。

2.80286 的内部结构

80286 CPU 的内部包括 4 个独立的执行部件:执行单元 EU、地址单元(Address Unit, AU)、指令单元(Instruction Unit, IU)和总线接口单元 BIU。这 4 个独立的部件每个都可以同其他部件异步并行操作,从而加快了 CPU 的处理速度。80286 的内部结构如图 3.20 所示。

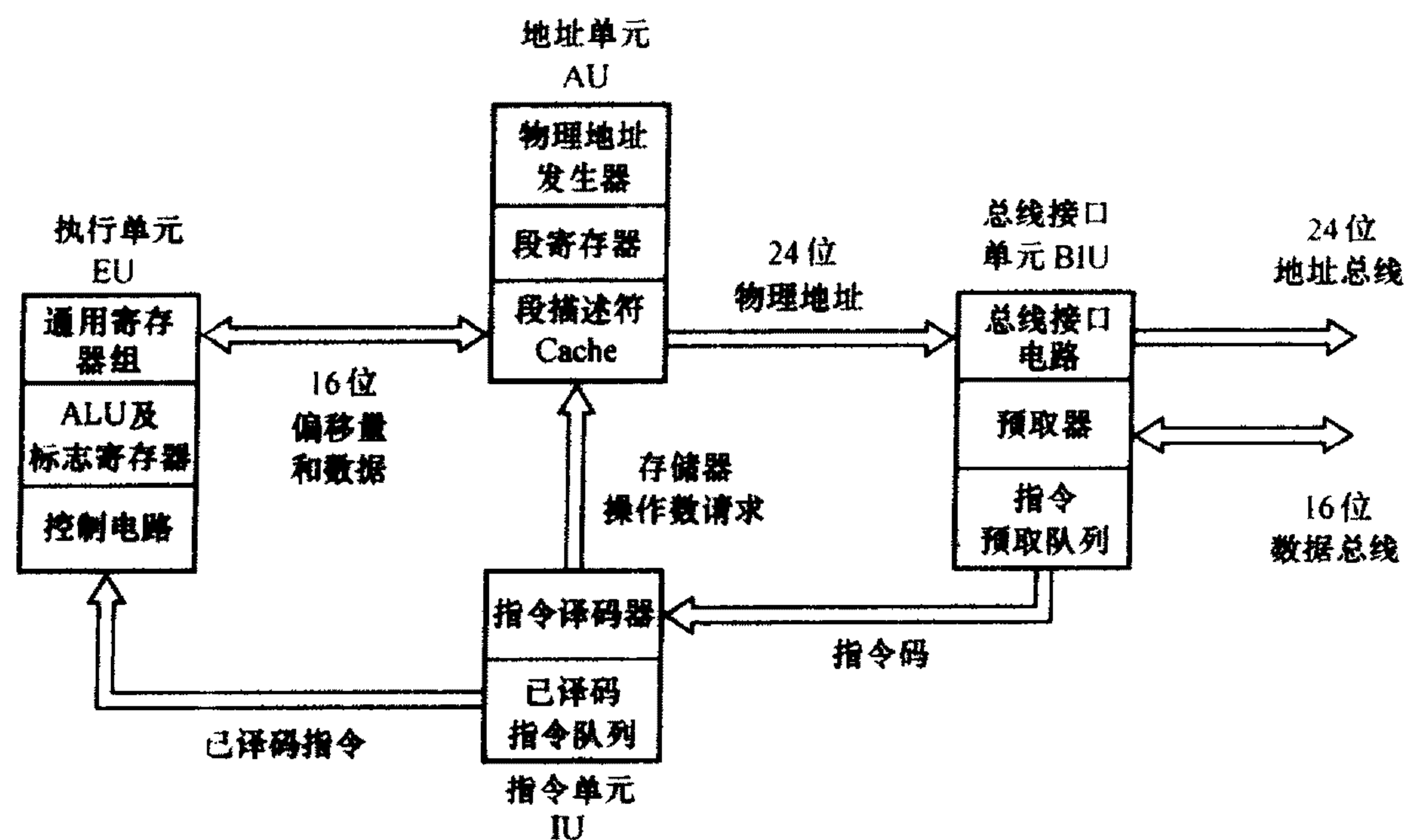


图 3.20 80286 内部结构示意图

1) 总线接口单元(BIU)

80286 BIU 的功能同 8086 中的一样,负责处理 CPU 与系统总线之间的数据传送,包括总线接口电路、预取器和 6 个字节的指令预取队列。当 CPU 不使用总线进行操作数存取时,BIU 的预取器从存储器中预取下一个要执行的指令存入预取队列中,只要队列有两个字节空闲,就可以预取。在遇到转移、子程序调用类指令时,将会使预取队列清零,并从转移的目标地址处开始预取指令。

2) 指令单元(IU)

IU 包括指令译码器和已译码指令队列。它负责将指令预取队列中的指令取出,送入指

令译码器。译码器将指令码译码为 69 位的内部码形式,然后存入已译码指令队列。已译码指令队列的容量为 69×3 位,即可同时保存 3 条被译码指令的内部码。

3) 执行单元(EU)

该单元与 8086CPU 中的 EU 大致相同,同样包括算术逻辑单元 ALU、通用寄存器组、标志寄存器及控制电路等。标志寄存器与 8086 相比增加了两个标志 IOPL 和 NT,其含义及标志寄存器中的标志位参见 3.5.2 小节的相关内容。控制电路根据指令的要求产生执行指令所需的控制信号,然后送入 EU 及其他部件,以完成指令的执行。通用寄存器组同样用来暂存运算的数据及中间结果,ALU 负责算术和逻辑运算,标志寄存器存放运算结果的特征。

4) 地址单元(AU)

AU 是 80286 的地址管理单元,包括物理地址发生器、段寄存器、段描述符 Cache(高速缓存器)等。

当 80286 运行在实地址模式时,AU 负责将段地址和偏移地址组合成 20 位的物理地址。

当 80286 运行在保护虚地址模式时,AU 在每次对存储器的存取操作时,都需做许可性检查和当前任务的段限制检查,以测试本次存取操作是否违反了存储器的保护机制。若操作合法,AU 就将逻辑地址或虚拟地址转换为 BIU 需要的物理地址。设置段描述符 Cache 是为了实现对存储器存取操作的保护功能,并加快逻辑地址向物理地址的转换。

3.5.2 80386 微处理器

1985 年 10 月,Intel 公司推出了与 8088/8086/80286 相兼容的高性能的 32 位 80386,它是为满足高性能的应用领域与多用户、多任务操作系统的需要而设计的。它的出品标志着微处理器自此从 16 位迈入了 32 位时代。

80386 采用先进的高速 CHMOS - III 工艺,既具有 HMOS 的高性能特点,又具有 CMOS 的低功耗的特点。该芯片集成了 27.5 万个晶体管,对外引脚 132 根,采用陶瓷网格阵列(PGA)封装,最高工作频率 16 MHz。

1. 80386 CPU 的主要特性

① 采用全 32 位结构,即它的内部寄存器、算术逻辑单元 ALU、数据总线以及地址总线均为 32 位,故能寻址的物理空间为 $2^{32} = 4 \text{ GB}$ 。

② 提供 32 位外部总线接口,最大数据传输率为 32 MB/s,具有自动切换数据总线宽度的功能。CPU 读/写数据的宽度可以在 32 位到 16 位之间自由进行切换。

③ 具有片内集成的存储器管理部件 MMU,可支持虚拟存储和特权保护,虚拟存储器空间可达 64 TB(2^{46} B)。存储器按段组织,每段最大 4 000 MB,因此 64 TB 虚拟存储空间允许每个任务可拥有多达 16 384 个段。存储保护机构采用四级特权层,可选择片内分页单元。内

部具有多任务机构,能快速完成任务的切换。

④ 具有三种工作方式,实地址方式、保护方式和虚拟 8086 方式。实地址方式和虚拟 8086 方式与 8086 相同,已有的 8088/8086 软件不加修改就能在 80386 的这两种方式下运行;保护方式可支持虚拟存储、保护和多任务,包括了 80286 的保护方式功能。

⑤ 采用了比 8086 更先进的流水线结构,使其能高效地、并行地完成取指、译码、执行和存储管理功能。它具有增强的指令预取队列,能预取指令并进行内部指令排队。取指和译码操作均由流水线承担,处理器执行指令不需等待。其指令预取队列从 80286 的 6 B 增加到 16 B。

2. 80386 内部基本结构

80386 内部结构如图 3.21 所示。它由三大部分组成:总线接口单元(BIU)、中央处理单元(CPU)和存储器管理单元(MMU)。

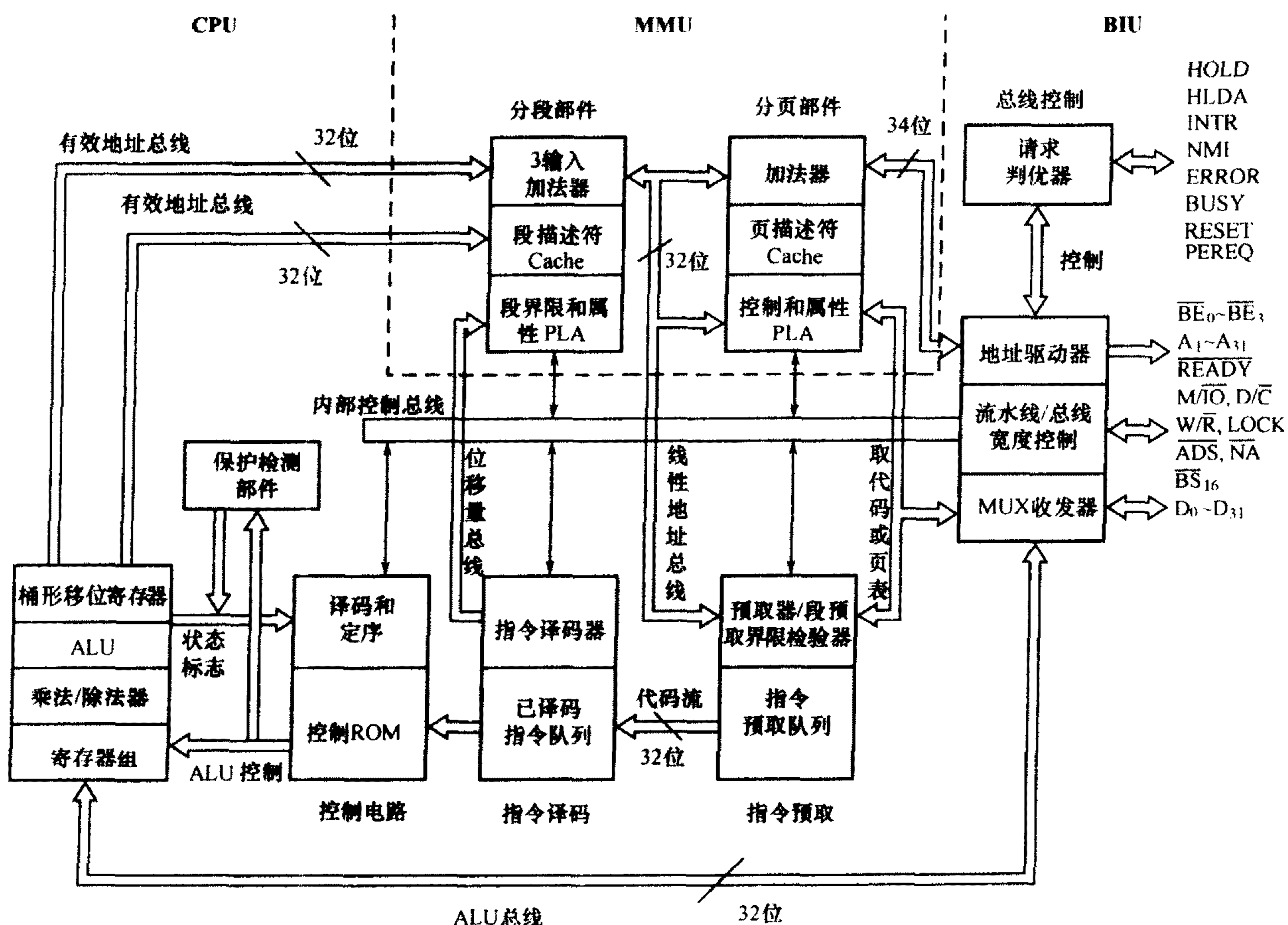


图 3.21 80386 微处理器内部结构

1) BIU

BIU 负责与存储器和 I/O 接口传送数据,其功能是产生访问存储器和 I/O 端口所必需的地址、数据和命令信号。其作用与 8088/8086 中的 BIU 作用相当。由于总线数据传送与总线地址形成可同时进行,所以 80386 的总线周期只用 2 个时钟。平常没有其他总线请求时,BIU 将下条指令自动送到指令预取队列。

2) CPU

CPU 包括指令预取单元、指令译码单元和执行单元三部分。

① 指令预取单元 (Instruction Prefetch Unit, IPU) 负责从存储器取出指令,放到一个 16 字节的指令队列中。它管理一个线性地址指针和一个段预取界限,负责段预取界限的检验。它把预取总线周期通过分页部件发给总线接口。每当预取队列不满或发生控制转移时,就向 BIU 发一个取指请求。指令预取的优先级别低于数据传送等总线操作。因此,绝大部分情况下是利用总线空闲时间预取指令。指令预取队列存放着从存储器取出的未经译码的指令。

② 指令译码单元 (Instruction Decode Unit, IDU) 从指令预取单元之中取出指令,进行译码。译码后的可执行指令放入已译码指令队列中,以备执行部件执行。每当已译码指令队列中有空闲时,就从预取队列中取出指令并译码。

③ 执行单元 EU 包括 8 个 32 位的寄存器组、32 位的算术逻辑单元 (ALU)、一个 64 位桶形移位寄存器和一个乘法/除法器。桶形移位寄存器能有效地实现移位、循环移位和位操作,被广泛地用于乘法及其他操作中。它可以在一个时钟周期内实现 64 位同时移位,也可对任何一种数据类型移任意位数。桶形移位寄存器与 ALU 并行操作,可加速乘法、除法、置位、移位和循环移位操作。

3) 存储器管理单元

存储器管理单元 MMU 由分段部件和分页部件组成。

① 分段部件的作用是应执行部件的请求,把逻辑地址转换成线性地址。在完成地址转换的同时还要执行总线周期的分段合法性检验。该部件可以实现任务之间的隔离,也可以实现指令和数据区的再定位。

② 分页部件的作用是把由分段部件或代码预取单元产生的线性地址转换成物理地址,并且要检验访问是否与页属性相符合。为了加快线性地址到物理地址的转换速度,80386 内设有一个页描述符 Cache,也称转换旁视缓冲存储器 (Translation Lookaside Buffer, TLB),其中可以存储 32 项页描述符,使得在地址转换期间,大多数情况下不需要到内存中查页目录表和页表。试验证明 TLB 的命中率可达 98%。对于在 TLB 内没有命中的地址转换,80386 设有硬件查表功能,从而缓解了因查表引起的速度下降问题。

3. 80386 的内部寄存器组

80386 共有 34 个内部寄存器(如图 3.22 所示),包括通用寄存器组、指令指针和标志寄

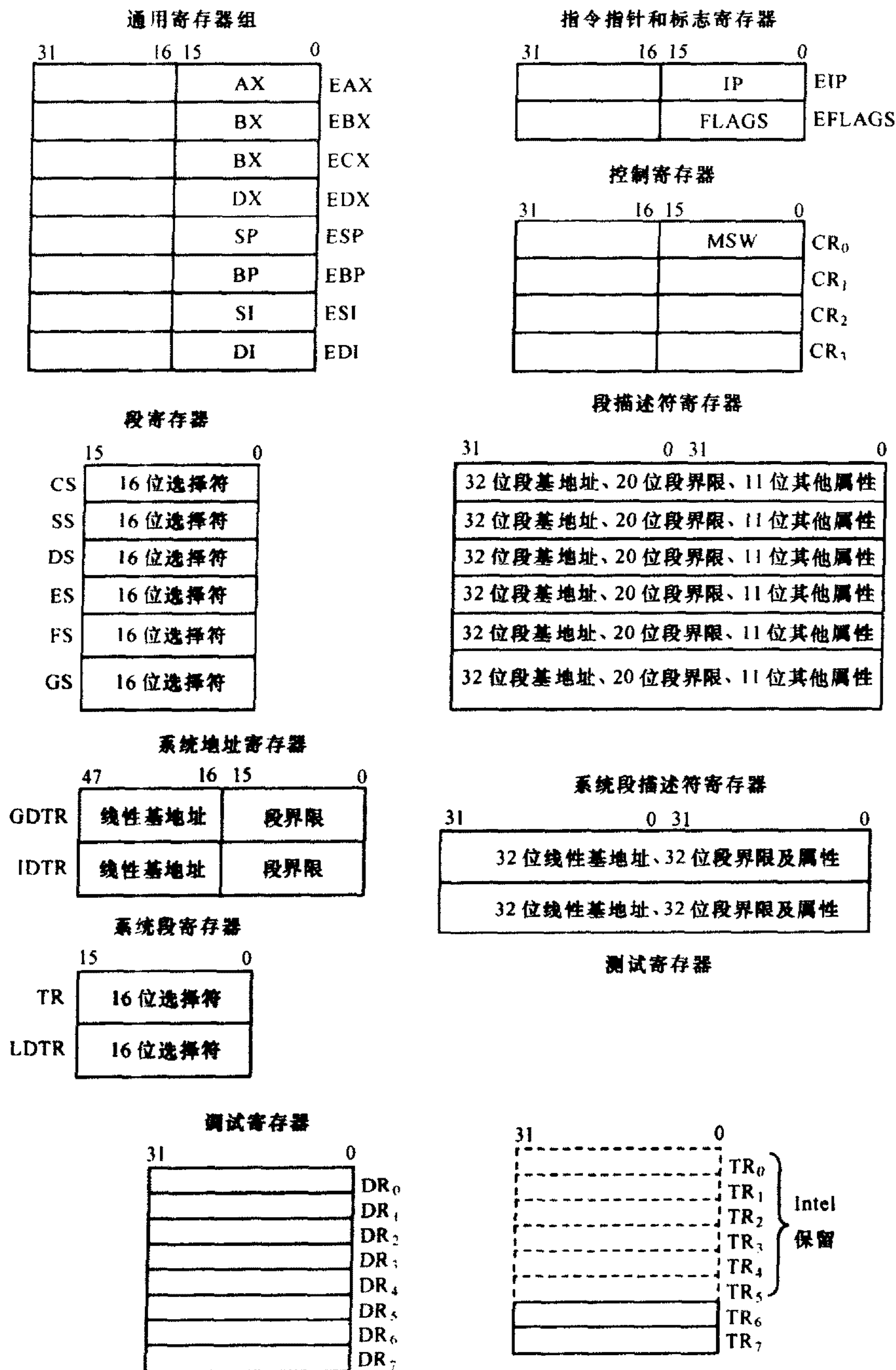


图 3.22 80386 的寄存器结构

1) 通用寄存器组

2) 指令指针和标志寄存器

VM 虚拟 8086 方式 (Virtual 8086 Mode, b_{17})。若 $VM = 1$, 处理器工作于虚拟 8086 方式; 若 $VM = 0$, 处理器工作于一般的保护方式。

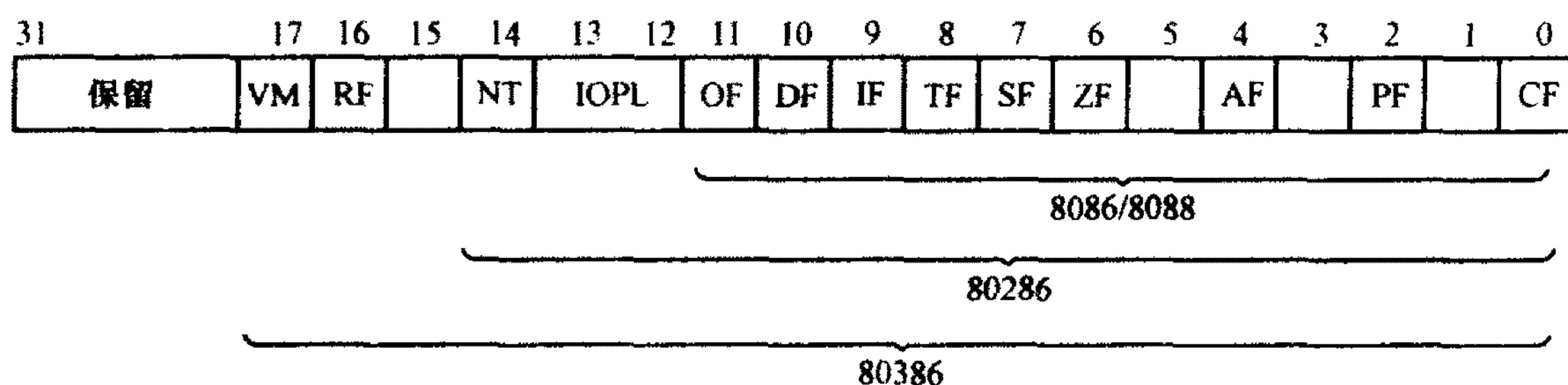


图 3.23 80386 的标志寄存器

3) 段寄存器

80386 内存单元的地址仍由段基地址和段内偏移地址组成。段内偏移地址为 32 位,由各种寻址方式确定。段基地址也是 32 位,但其不能像 8086 那样直接由 16 位段寄存器左移 4 位而得。而是根据段寄存器的内容,通过一定的转换才能得出。因此,为了描述每个段的

性质,80386 内部的每一个段寄存器都对应着一个与之相联系的段描述符寄存器,用来描述一个段的段基地址、段界限和段的属性。每个段描述符寄存器有 64 位,其中 32 位为段基地址,另外 32 位为段界限(本段的实际长度)和必要的属性。段描述符寄存器对程序员是不可见的,程序员通过 6 个段寄存器间接地对段描述符寄存器进行控制。在保护方式(多任务方式)下,6 个 16 位的段寄存器也称为段选择符,即段寄存器中存放的是某一个段的选择符。当用户将某一选择符装入一个段寄存器时,控制单元会自动用段寄存器中的值作为索引从段描述符表中取出一个 8 B 的描述符,装入到与该段寄存器相应的 64 位描述符寄存器中。这个过程是由硬件自动完成的。

一旦段描述符被装入段描述符寄存器中,在以后访问存储器时,就可使用与所指定的段寄存器相应的段描述符寄存器中的段基地址来计算线性地址,而不必每次访问时都去查找段描述符表。

4) 控制寄存器

80386 有 4 个 32 位控制寄存器(CR_0 、 CR_1 、 CR_2 和 CR_3),它们的作用是保存全局性的机器状态,其中 CR_0 的格式如图 3.24 所示。

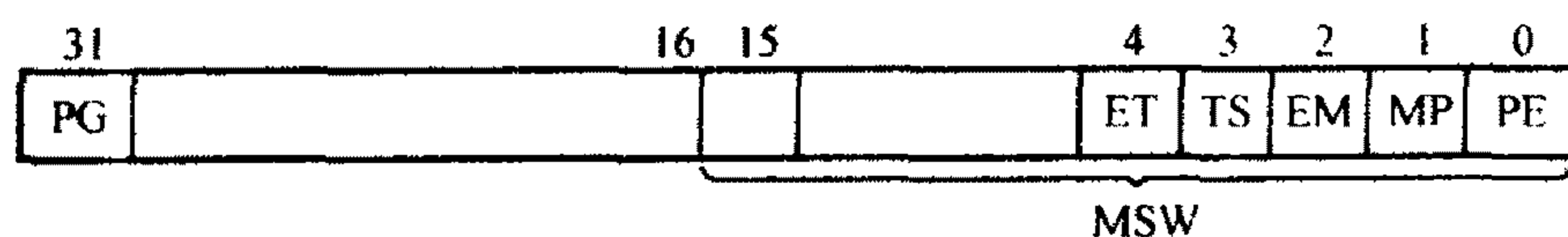


图 3.24 控制寄存器 CR_0 的结构

CR_0 的低 16 位称为机器状态字 MSW,其中:

PE 保护允许位。进入保护方式时 $PE = 1$ 。除复位外,其不能被清除。实方式时 $PE = 0$ 。

MP 监视协处理器位。当协处理器工作时 $MP = 1$,否则 $MP = 0$ 。

EM 仿真协处理器位。当 $MP = 0$,且 $EM = 1$ 时,表示要用软件来仿真协处理器功能。

TS 任务切换位。当两任务切换时, $TS = 1$,此时不允许协处理器工作;当两任务之间切换完成后, $TS = 0$ 。

ET 协处理器类型位。系统配接 80387 时, $ET = 1$;配接 80287 时, $ET = 0$ 。

PG 页式管理允许位。 $PG = 1$ 表示启用芯片内部的页式管理系统,否则 $PG = 0$ 。

CR_1 由 Intel 公司保留。

CR_2 存放引起页故障的线性地址。注意,仅当 CR_0 的 $PG = 1$ 时,才使用 CR_2 。

CR_3 存放当前任务的页目录基地址。同样,仅当 CR_0 的 $PG = 1$ 时,才使用 CR_3 。

5) 系统地址(段)寄存器

80386 有 4 个系统地址(段)寄存器,用来存储操作系统需要的保护信息和地址转换表信

息,定义目前正在执行任务的环境、地址空间和中断向量空间。

GDTR 48 位全局描述符表寄存器,用于保存全局描述符表的 32 位基地址和全局描述符表的 16 位段界限(全局描述符表最大为 2^{16} B,共 $2^{16}/8 = 8$ K 个全局描述符)。

IDTR 48 位中断描述符表寄存器,用于保存中断描述符表的 32 位基地址和中断描述符表的 16 位段界限(中断描述符表最大为 2^{16} B,共 $2^{16}/8 = 8$ K 个中断描述符)。

TR 16 位任务状态寄存器,用于保存任务状态段(TSS)的 16 位选择符。与 LDTR 相同,一旦 16 位的选择符放入 TR,CPU 会自动将该选择符所指定的任务描述符装入 64 位的任务描述符寄存器中。

LDTR 16 位局部描述符表寄存器,用于保存局部描述符表的选择符。一旦 16 位的选择符放入 LDTR,CPU 会自动将选择符所指定的局部描述符装入 64 位的局部描述符寄存器中。

LDTR 和 TR 寄存器由 16 位选择字段和 64 位描述符寄存器组成,用来指定局部描述符表和任务状态段 TSS 在物理存储器中的位置和大小。64 位描述符寄存器是自动装入的,程序员不可见。LDTR 与 TR 只能在保护方式下使用,程序只能访问 16 位选择符寄存器。

6) 调试寄存器

80386 设有 8 个 32 位调试寄存器 $DR_0 \sim DR_7$,它们为调试提供了硬件支持。其中, $DR_0 \sim DR_3$ 是 4 个保存线性断点地址的寄存器; DR_4 、 DR_5 为备用寄存器; DR_6 为调试状态寄存器,通过该寄存器的内容可以检测异常,并进入异常处理程序或禁止进入异常处理程序; DR_7 为调试控制寄存器,用来规定断点字段的长度、断点访问类型、“允许”断点和“允许”所选择的调试条件。

7) 测试寄存器

80386 设置了 8 个 32 位的测试寄存器 $TR_0 \sim TR_7$,其中 $TR_0 \sim TR_5$ 由 Intel 公司保留,用户只能访问 TR_6 、 TR_7 。它们用于控制对 TLB 中的 RAM 和 CAM 相联存储器的测试。 TR_6 是测试控制寄存器, TR_7 是测试状态寄存器,保存测试结果的状态。

4. 外部引脚及其功能

80386 共有 132 根引脚,使用 PGA 封装技术。它对外直接提供了独立的 32 位地址总线和 32 位数据总线。80386 能在 2 个时钟周期内完成 32 位数据传送,在 33 MHz 工作频率下,其传送速率为 66 MB/s。

80386 引脚定义如下:

CLK2 两倍时钟输入信号。该信号与 80386 时钟信号同步输入,在 80386 内部二分频后产生指令执行时钟 CLK。每个 CLK 由两个 CLK2 时钟周期组成,分别称其为相 1 和相 2。

$D_0 \sim D_{31}$ 数据总线信号,双向,三态。一次可传送 8、16、32 位数据。

$A_2 \sim A_{31}$ 地址总线信号输出,三态。和 $\overline{BE}_0 \sim \overline{BE}_3$ 相结合可起到 32 位地址作用。

$\overline{\text{BE}}_0 \sim \overline{\text{BE}}_3$ 字节选通输出信号,每条线控制选通一个字节; $\overline{\text{BE}}_0 \sim \overline{\text{BE}}_3$ 分别对应选通 $D_0 \sim D_7$ 、 $D_8 \sim D_{15}$ 、 $D_{16} \sim D_{23}$ 与 $D_{24} \sim D_{31}$,相当于存储器分为4个存储体,与 $A_2 \sim A_{31}$ 结合可寻址 $2^{30} \times 2^2 = 4 \text{ G}$ 个内存单元。

W/R 读/写控制,输出信号。

D/C 数据/控制输出信号。表示是数据传送周期还是控制周期。

M/IO 存储器与 I/O 选择信号,输出。

LOCK 总线锁定输出信号。

ADS 地址状态,三态输出信号。表示总线周期中地址信号有效。

NA 下一地址请求,输入信号。允许地址流水线操作,即当前周期发下一总线周期地址的状态信号。

BS₁₆ 总线宽度为16的输入信号。

READY 准备就绪,输入信号。表示当前总线周期已完成。

HOLD 总线请求保持,输入。

HLDA 总线响应保持,输出。

PEREQ 处理器扩展请求,输入。表示80387要求80386控制它们与存储器之间的信息传送。

BUSY 协处理器忙,输入。

ERROR 协处理器出错,输入。

NMI 不可屏蔽中断请求信号,输入。

INTR 可屏蔽中断请求信号,输入。

RESET 复位信号。

5. 80386 的工作模式

80386可工作于实地址模式、虚拟8086模式及保护虚地址模式。

1) 实地址模式

当80386加电或复位后,就进入实地址工作模式。此时的80386就相当于一个高速的8086。物理地址的形成也与8086一样,即将段寄存器内容左移4位与偏移地址相加得到,寻址空间为1MB,只有地址线 $A_2 \sim A_{19}$ 、 $\overline{\text{BE}}_0 \sim \overline{\text{BE}}_3$ 是有效的。而 $A_{20} \sim A_{31}$ 总是低电平。但有一点例外,就是在复位后,执行第一条段间转移或调用指令前,所有访问代码段的总线周期的地址 $A_{20} \sim A_{31}$ 输出总是高电平,以保证执行高端内存引导ROM中的指令。在实地址模式下,段的大小为64KB,因此32位的有效地址必须小于0000FFFFH。此模式下保留了两个固定的存储区域,它们是专用的:

① 中断向量表区 00000H~003FFH,在1KB的存储空间中保留了256个中断服务程

序的入口地址,每个入口地址占用 4 个字节,与 8086 一样;

② 系统初始化区 FFFFFFF0H ~ FFFFFFFFH,存放 ROM 引导程序。

实地址模式下的物理地址生成如图 3.25 所示。

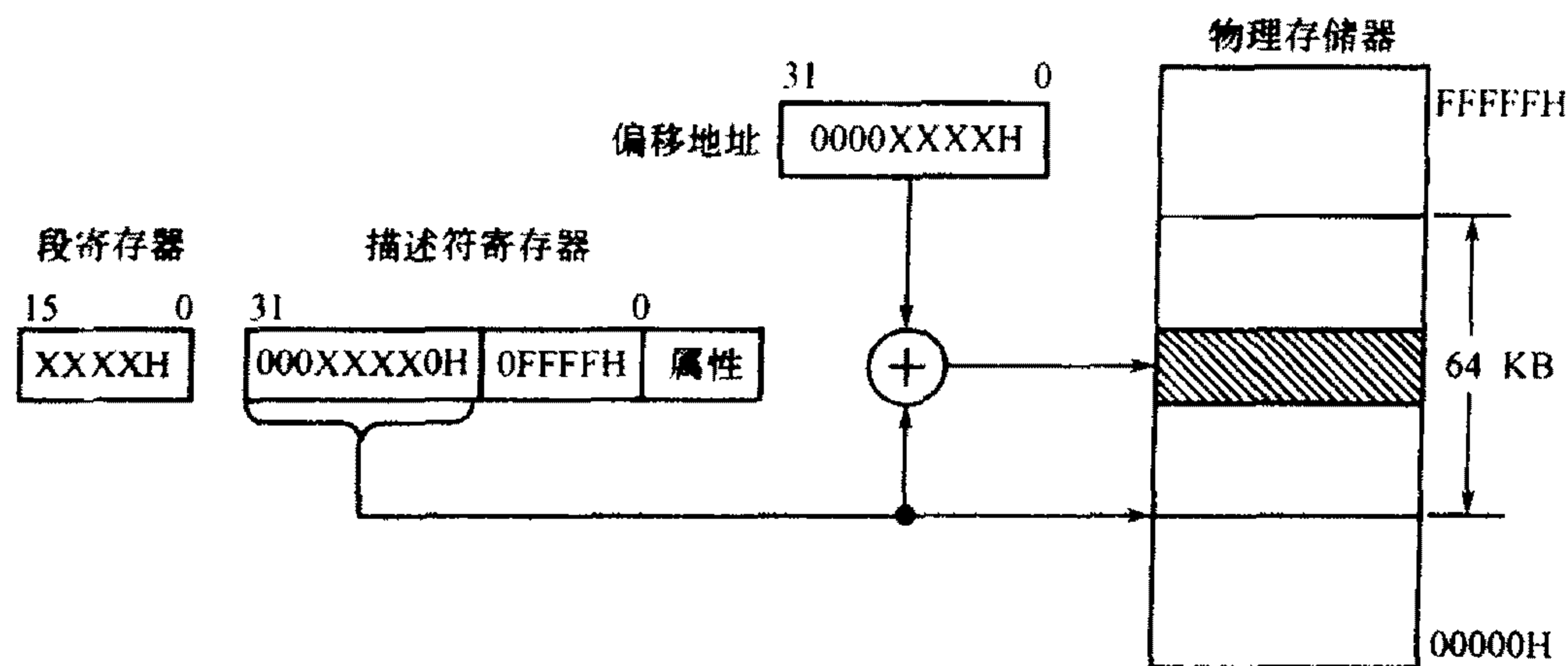


图 3.25 实模式下的地址转换

2) 虚拟 8086 模式

此模式是由软件(操作系统)来模拟一个虚拟的 8086 机。它是在实地址模式下运行 8086 应用程序的同时,利用 80386 的虚拟保护机构实现多个用户程序同时运行的功能,而且这些程序都能得到保护,就像每个用户都自己拥有一台完整的计算机一样。

3) 保护虚地址模式

当 80386 工作在保护方式时,其访问的线性地址空间可达 4 GB(2^{32}),而且允许运行几乎不受存储空间限制的虚拟存储器程序。用户逻辑地址空间,即虚拟存储器地址空间可达 64 TB(2^{46})。在此模式下,80386 提供了复杂的存储管理和硬件辅助的保护机构,且可运行现有 8088/8086/80286 的所有软件。它还增加了支持多任务操作系统的特别优化的指令。

实际上,64 TB 的虚拟地址空间是由磁盘等外部存储器的支持下实现的。它与 CPU 的存储器管理部件、48 位和 16 位的描述符表寄存器等有直接关系。编写程序时,程序可以放在磁盘存储器上,但在执行时,必须把程序加载到物理存储器中。存储器管理的任务就是要将 46 位虚拟地址转换成 32 位物理地址。

(1) 保护模式的地址转换

在保护模式下,某个段寄存器中的内容不再是段的基地址,32 位的段基地址是存放在一个段描述符表中,而段寄存器的内容作为选择符,即作为段描述符表的索引,用它来从表中取出相应的段描述符(包括 32 位段基地址、段界限和访问权等)。在这个地址转换过程中,由选择符的高 13 位作为偏移量,再以 CPU 内部预先初始化好的 GDTR 中的内容作为基

地址,就可获得相应的描述符。该描述符将存入描述符寄存器中。描述符中的段基地址(32位)与指令给出的32位偏移地址相加得到线性地址,再通过分页部件进行转换,最后得到物理地址。如果不分页,线性地址就等于物理地址。保护模式下的地址转换如图3.26所示。

(2) 描述符表

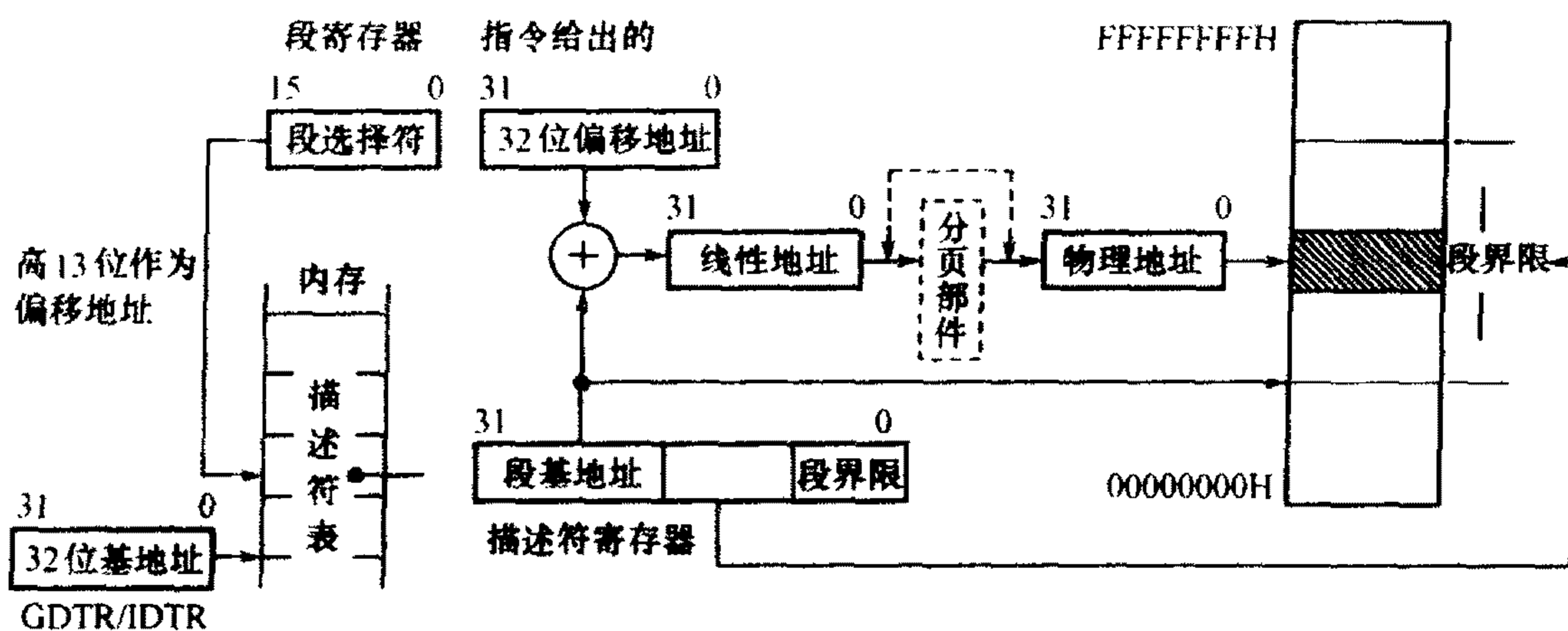


图 3.26 保护模式下的地址转换

① 段选择符 在保护虚地址方式下,段寄存器就成为一个选择符,它由3个字段组成,如图3.27所示。

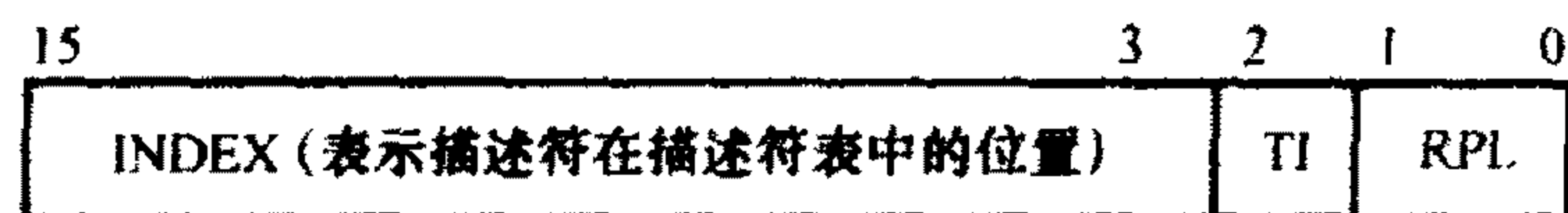


图 3.27 放在段寄存器中的选择符的格式

INDEX 13位的索引值,给出要选择的描述符在描述符表中的位置,或描述符在表中的序号。INDEX加上32位的描述符表基地址(在GDTR/LDTR中)就得到所选择的描述符的物理地址。最大可选择8K个全局描述符和8K个局部描述符,共16K个描述符。

TI 描述符表指示器, TI=0,表示指向全局描述符表GDT; TI=1,表示指向局部描述符表LDT。

RPL 选择器特权级,定义当前请求的特权层级别,特权级为0~3级,0级最高,3级最低。

② 段描述符 用段选择器从描述符表中所选择的对象称为段描述符。段描述符包含了一个存储分段的所有信息。其中包括段的线性基地址和此段的界限(大小),还包括段的一些属性。这些属性包括:此段的保护等级,读、写、执行特权和保护特权级别,操作数的默认长度(16或32位),段的类型和段的粒度(即段的长度单位)。

每个段描述符有8B,其中段基地址32位(4B),段界限字段20位,还有12位定义了段

的属性信息,其格式如图 3.28 所示。

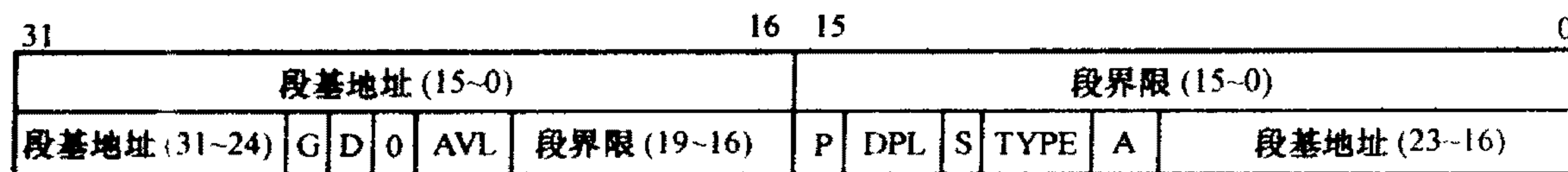


图 3.28 段描述符的格式

段描述符的 32 位基地址规定了存储分段在线性地址空间中的基地址。

③ 段界限字段指定了该存储分段的长度。

④ 粒度位 G 确定分段的最大地址空间 G = 0 时,段长是以字节为单位,此时,该分段的最大地址空间是 1 MB;G = 1 时,段长是以页为单位,此时,该分段的最大地址空间是 4 GB。

⑤ D 位指示了操作数和有效地址的缺省长度 D = 1,使用 32 位操作数和 32 位寻址方式;D = 0,使用 16 位操作数和 16 位寻址方式。

⑥ P 位表示该存储分段是否在内存 P = 1,表示该分段已在内存;P = 0,表示该分段在外存硬盘交换区。

80386 把段分成两类:系统段和非系统段。系统段描述操作系统的系统表、任务和门的信息;非系统段就是代码段和数据段。段描述符中的 S、TYPE 和 A 字段有三种组合方式:数据段或堆栈段(S = 1 且 TYPE 字段的 E = 0)、代码段(S = 1 且 TYPE 字段的 E = 1)、系统段(S = 0)。

3.5.3 Pentium 4 微处理器

1. Pentium 处理器的发展

80386 的后继者是 80486,其结构基本上是将 80386 微处理器、80387 数字协处理器和高速缓存集成在一块芯片上的产品。而从 80486 开始一直到现在的 Pentium 处理器,从程序员的观点来看,其主要的结构则都仅仅是在前一代产品的基础上进行了一些新的改进,如增加了多媒体处理部件、集成了更大的缓存,将二级高速缓存(L2 Cache)也做到了 CPU 内部、采用了多级、多流水线的超标量结构等。其主要的程序特性,如支持多用户和多任务、具有硬件保护功能、支持分段分页虚拟存储等均与 80386 基本类似。下面简单介绍一下 Pentium 家族中的最新产品 Pentium 4 的技术特点。

Pentium 微处理器目前有 5 个不同版本(升级): Pentium、Pentium MMX、Pentium II、Pentium III 和 Pentium 4。其发展过程如图 3.29 所示。

Pentium 4 是最新的 IA - 32 体系结构的微处理器,也是第一款基于 Intel NetBurst 微体系结构(Intel Micro - architecture)的微处理器。Intel NetBurst 微体系结构是一种允许处理器运行在极高时钟速率,比以前的 IA - 32 体系结构具有更高性能级别的新型 32 位微体系结构。

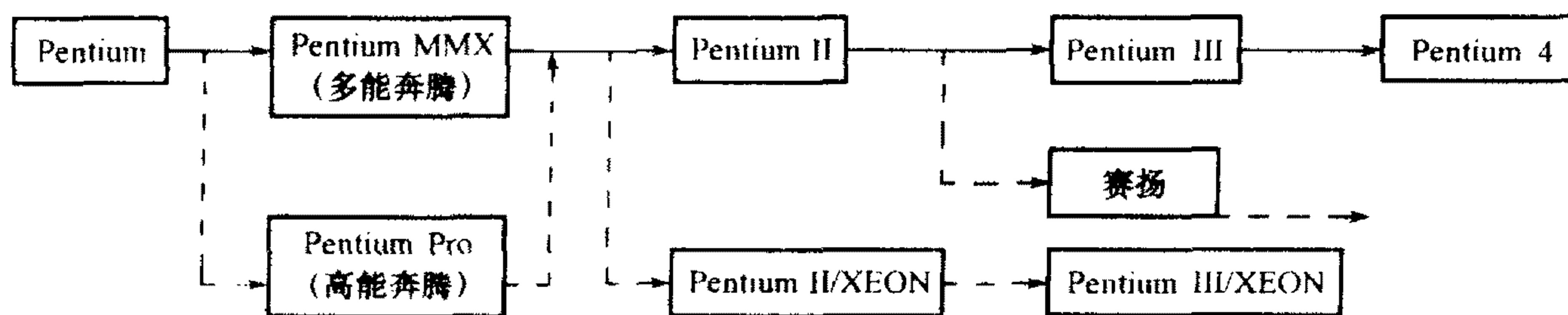


图 3.29 Pentium 微处理器的发展

采用 NetBurst 体系结构的 Pentium 4 处理器具有以下先进的特性：

1) 首次实现了 Intel NetBurst 微体系结构

① 快速执行引擎使处理器的算术逻辑单元执行速度达到了内核频率的两倍,从而实现了更高的执行吞吐量。

② 超长流水线技术使流水线深度比 Pentium III 增加了一倍,达到 20 级,显著提高了处理器性能和执行速度。

③ 创新的新型高速缓存子系统使指令执行更加有效。

④ 增强的动态执行结构可以对更多的指令进行转移预测处理(比 Pentium III 处理器多 3 倍),有效地避免因发生程序转移使流水线停顿的现象(因为一旦预测不正确,CPU 将不得不重新填充指令队列)。

2) 流式 SIMD(单指令多数据)扩展 2(SSE2)技术

① 扩展了 MMX 和 SSE 技术。新增加了 144 条多媒体处理指令,可用来支持: 128 位 SIMD 整数运算和 128 位 SIMD 双精度浮点运算。

② 进一步增强和加速了视频、语音、数据加密、图像和影像处理功能。

3) 400 MHz Intel NetBurst 微体系结构系统总线

① 提供了 3.2 GB/s 的吞吐率(比 Pentium III 快 3 倍)。

② 4 倍速的 100 MHz 可升级的总线时钟使有效速度达到 400 MHz。

③ 片断化事务,深度流水线化操作。

④ 128 B 输管线,每次存取 64 B

4) 与已有的为 IA-32 体系结构而编写的应用和操作系统完全兼容

目前已推出的 Pentium 4 处理器的主频分别为 1.30 GHz、1.40 GHz、1.50 GHz、1.60 GHz、1.70 GHz、1.80 GHz、2 GHz 的版本,到 2003 年初已推出 3.1 GHz 的版本。

Pentium 4 芯片的大小为 217 mm^2 ,内含 3 400 万(新版本为 4 200 万)个晶体管,采用 $0.18 \mu\text{m}$ 工艺制造。外部引脚数 423 根,其外形封装见图 3.30。Pentium 4 的功能框图如图 3.31 所示,其中:

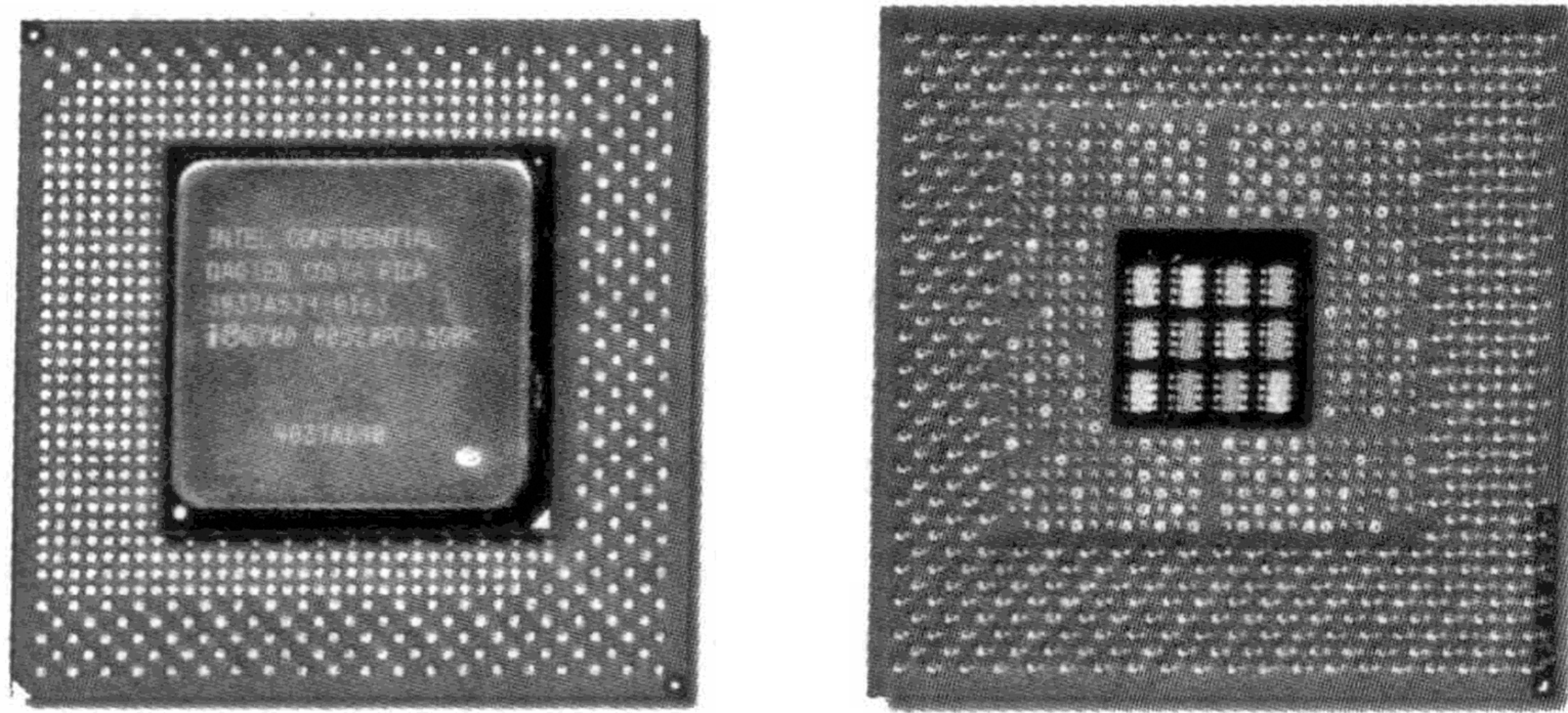


图 3.30 Pentium 4 微处理器的正面和反面

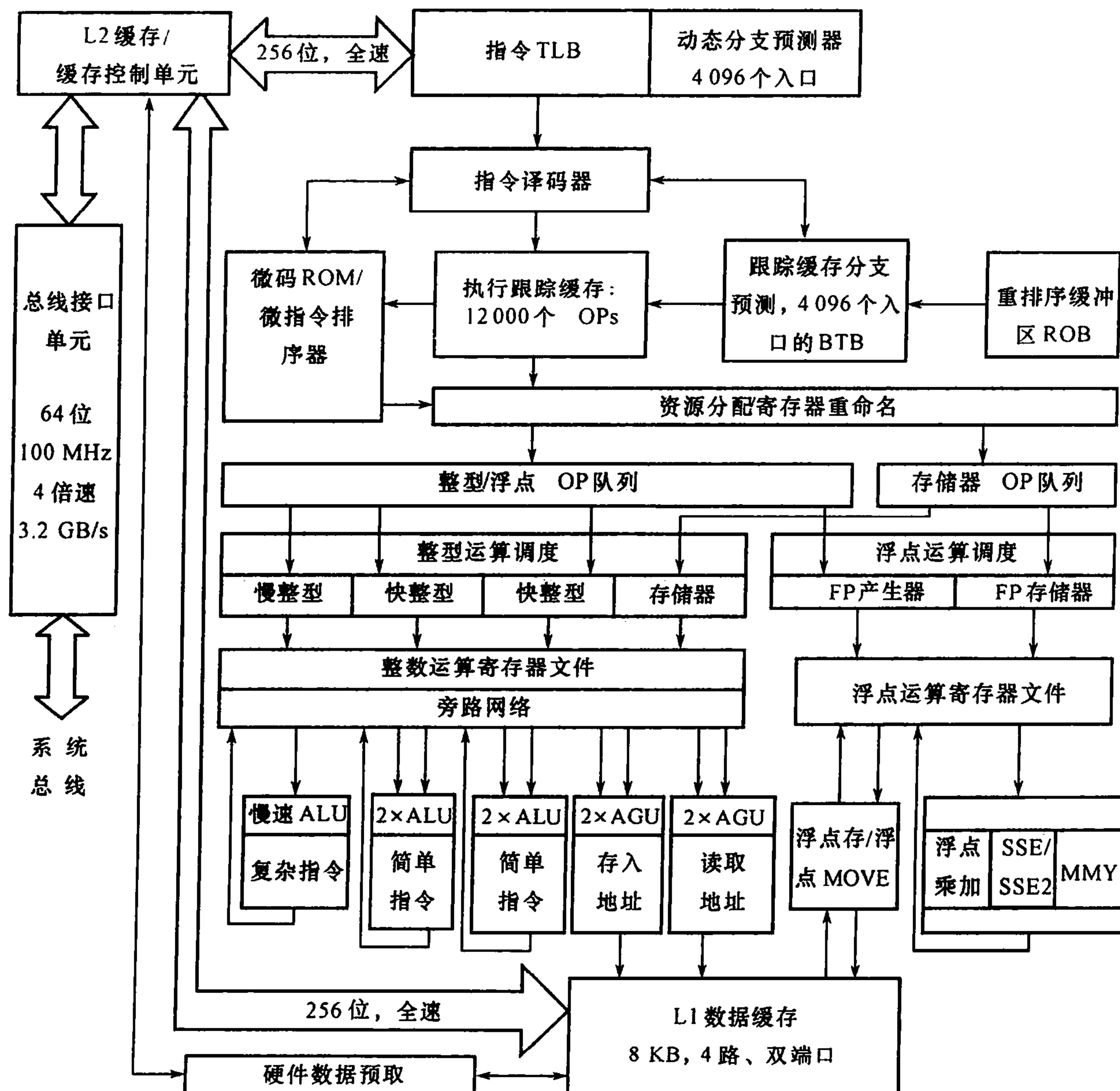


图 3.31 Pentium 4 微处理器的功能框图

BTB 分支目标缓冲区(Branch Target Buffer)。存放所预测分支的所有可能的目的地址。

μOP 微操作(Micro - Operation)。CPU 的执行单元能直接执行的指令称为微指令(或微命令),在接受微指令后执行单元所进行的操作称为微操作。一条指令可分解为一系列微指令的执行。与 X86 的变长指令集不同,微指令的长度是固定的,因此,很容易在执行流水线中进行处理。指令译码部件将 X86 指令转成一条或多条微指令,对于非常复杂的 X86 指令则会生成较多的微指令。例如,大多数 X86 指令会被编译成 1~2 条微指令;一些很简单的指令如 AND、OR、XOR、ADD 等,仅被编译成一条微指令;而 DIV、MUL 以及间接寻址运算则会生成较多的微指令;极为复杂的指令如三角函数等,则可能会生成上百条微指令。在现代超标量微处理器中,微指令存放在内部的一个“微码存储器(Micro Code ROM,微码 ROM)”中。

ALU 算术逻辑单元,即整数运算单元。数学运算如加、减、乘、除以及逻辑及移位运算,如 AND、OR、ASL、ROL 等指令都在算术逻辑单元中执行。在程序中,这些运算占了绝大多数,所以 ALU 的性能在很大程度上决定了系统的性能。

AGU 地址产生单元(Address Generation Unit)。AGU 负责在信息取出或存入时,决定正确的地址。一般程序很少用绝对寻址的方式,程序中进行数组操作时,通常用的是间接寻址,这会使 AGU 单元持续处于忙状态。

2. 流式 SIMD 扩展 2(SSE2)技术

Pentium 4 处理器的 SSE2 扩展对 MMX 技术和 SSE 扩展进行了增强。其中包括对紧缩型数据进行操作,使用 128 位宽的寄存器进行整数 SIMD 运算,增强了 SIMD 计算能力。新增加了对紧缩的双精度浮点数据类型和几种紧缩的 128 位整数类型的支持。这些新的数据类型允许紧缩的双精度、单精度浮点以及紧缩的整型数运算在 XMM 寄存器中进行。

新增加的 144 条 SIMD 指令包括浮点 SIMD 指令、整型 SIMD 指令、SIMD 浮点数和 SIMD 整型数互相转换的指令、在 XMM 和 MMX 寄存器之间的紧缩数据转换指令。新的浮点 SIMD 指令允许以紧缩的双精度浮点值进行运算(每个 XMM 寄存器存放 2 个双精度值)。SIMD 浮点的运算指令、单精度和双精度浮点格式均与二进制浮点算术运算的 IEEE 754 标准兼容。新的整型 SIMD 指令通过支持双字和四字的算术操作以及支持对紧缩的字节、字、双字、四字以及双四字的其他操作提供了灵活的极大动态范围的计算能力。SSE2 可同时使用以下类型的数据:

- ① 4 个单精度浮点数;
- ② 2 个双精度浮点数;
- ③ 16 个字节整数;
- ④ 8 个字整数;
- ⑤ 4 个双字整数;
- ⑥ 2 个四倍字整数;

⑦ 1 个 128 位长的整数。

丰富的数据类型和新增加的 SSE2 指令集大大提高了 Pentium 4 在多媒体应用领域的性能。

除了新的 128 位 SIMD 指令外, Pentium 4 也允许在 Pentium II 和 Pentium III 中的 68 条 SIMD 指令在 128 位的 XMM 寄存器中进行 128 位运算。这些增强的整数 SIMD 指令允许软件开发者开发具有更高性能的浮点和整数算法, 以及可以灵活地选用 XMM 寄存器或 MMX 寄存器编写 SIMD 代码。

为了加快处理速度以及增加高速缓存的利用率, SSE2 扩展提供了几条新的指令, 以允许程序员来控制数据的可缓存能力。这些指令具有使数据流入和流出寄存器而不破坏缓存的能力, 还具有在数据实际被使用前就预取的能力。

3. P6 系列的微体系结构

微体系结构是从 Pentium Pro 开始被引入到 IA-32 处理器的, 故通常将这种结构称为 P6 处理器微体系结构。P6 处理器微体系结构通过加入集成的 L2 缓存而使功能增强, 这个 L2 缓存称为高级传输缓存 (Advanced Transfer Cache)。这种微体系结构是一种 3 路超标量的流水线体系结构。3 路超标量是指使用并行处理技术的处理器, 平均每个时钟周期可执行 3 条指令的译码、分发和执行动作。为了控制这个指令流, P6 系列的处理器使用了一个非连接的 L2 级超流水线来支持乱序 (Out-Of-Order) 指令执行。图 3.32 所示为具有高级传输缓存的 P6 微体系结构流水线的概念框图。

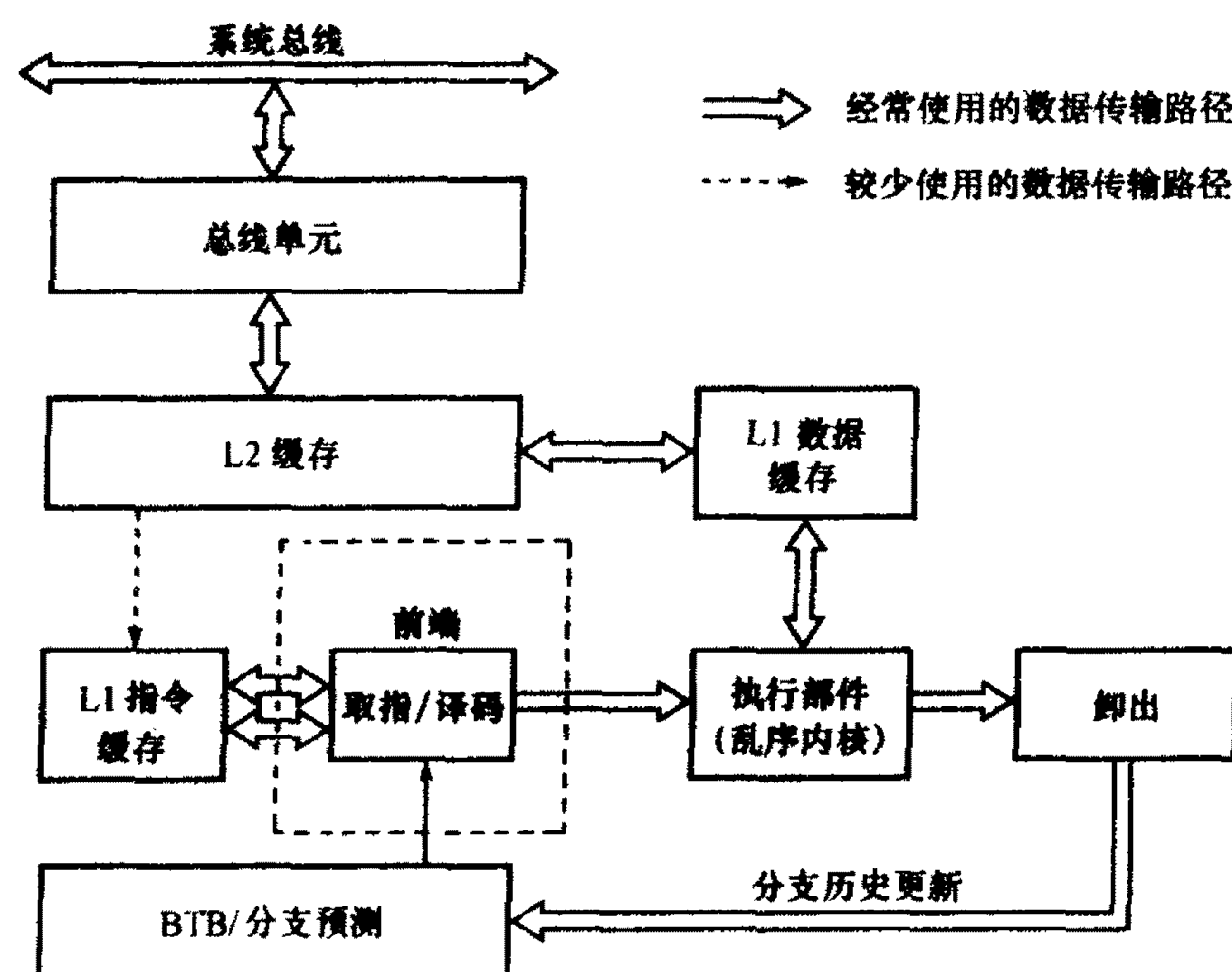


图 3.32 具有高级传输缓存的 P6 微体系结构

微体系结构的流水线分为4个部分(L1和L2缓存、前端、执行部件(乱序内核)以及卸出)。指令和数据通过总线单元馈送到这些部件。

为了保证稳定地为执行指令流水线供应指令和数据,P6微体系结构配置了两级缓存。第一级提供8KB的指令缓存和8KB的数据缓存,它们都与流水线紧密地结合在一起。第二级为256KB、512KB或1MB的静态缓存(RAM),它通过一个全速的64位缓存总线与处理器内核相连。

P6微体系结构的核心就是创新的乱序执行机制,称为动态执行(Dynamic Execution)。动态执行包括三个数据处理的概念:深度分支预测(Deep Branch Prediction)、动态数据流分析(Dynamic Data Flow Analysis)、投机执行(Speculative Execution)。

分支预测是一种能够提供高性能流水线微结构的现代技术。它允许处理器在分支之前对可能要执行的指令译码,以保持指令流水线满负荷运行。P6处理器系列实现了高度优化的分支预测算法,通过多级分支、过程调用和返回来预测指令流的方向。

动态数据流分析包括通过处理器对数据流的实时分析来决定数据与寄存器的依赖关系,并检测乱序指令执行的时机。

乱序执行核心能够同时监视许多指令,并以使处理器多个执行单元的使用达到最优化的顺序来执行这些指令。乱序执行使处理器的执行单元保持忙,甚至当缓存未命中或指令之间出现了数据依赖关系也是如此。

推测执行是指处理器在条件转移的目标未知时就提前执行那些仅在条件转移发生后才能够确定执行的指令的能力,并且最终能够以原始指令流相同的顺序提交结果。为使推测执行成为可能,P6处理器的微体系结构把指令的调度和执行与最终结果的提交分离开来。处理器的乱序执行核心利用数据流分析技术,预先执行指令池中所有可能要被执行的指令,并把不同的执行结果存在临时寄存器中。接着,卸出单元顺序地搜索指令池,查找那些已被执行完的、与其他指令或尚无结果的转移预测不再有数据依赖关系的指令。若找到这样的指令,卸出单元就把这些指令的执行结果按原来指令的顺序提交到存储器或寄存器,然后把这些指令从指令池中卸出。

通过分支预测、动态数据流分析和推测执行三者的有机结合,P6微体系结构的动态执行能力消除了指令执行时在传统的取指和执行阶段之间其指令序列必须是线性顺序的限制。这样就能使处理器不间断地进行指令译码(甚至当指令流中有多级分支跳转时也是如此);分支预测和先进的译码器能保证指令流水线始终处于满的状态。乱序推测执行引擎能够利用处理器的6个执行单元来并行地执行指令。最后,指令的结果能够按照原来程序中指令的顺序被提交,以保证数据的完整性和程序的一致性。

4. NetBurst 微体系结构

1) NetBurst 微体系结构的重要特性

(1) 快速执行引擎(Rapid Execution Engine)

① 包括两组 2 倍速的 ALU 及两组 2 倍速的 AGU(其中一组用于存储地址生成,一组用于读取地址生成),另外还有一个用于执行复杂运算指令的慢速 ALU。

② 4 个 ALU 和 AGU 均可在一个时钟周期内执行两次基本的整型运算,因此执行速度为 CPU 主频的两倍。

③ 如果遇到快速执行引擎不能执行的复杂指令,将由慢速 ALU 执行。

注意,如果复杂指令太多的话,会在很大程度上影响微处理器的性能发挥。但根据统计,程序中使用到的指令 95% 以上都是一些简单的指令,如 ADD、SUB、OR、XOR、AND 等,因此快速执行引擎处理单元能较好地发挥出其高速运算的性能。

(2) 超级流水线技术(Hyper Pipelined Technology)

Pentium 4 具有 20 级的超长流水线。相比之下 Pentium III 的流水线仅为 10 级。超长流水线使 Pentium 4 中的所有部件都能以极高的时钟频率全速运行。但要注意,超长流水线也有其弱点,如果在流水线的最后执行步骤,程序分支跳转到了预测单元并未预测到的地方,则整个流水线中的内容需清除并重新进行填充。在这种情况下,流水线越长,重新让流水线填满的时间也会越长,造成的预测失败延迟就越大。

(3) 高级动态执行

① 极深的乱序推测执行引擎(Out-of-order Branch Prediction Engine)。

- 同时执行的指令可高达 126 条;
- 流水线可同时执行 48 个读取操作和 24 个存储操作。

② 增强的分支预测能力。

- 降低了超长流水线在预测失败时必须清除整个流水线内容造成的延迟;
- 先进的分支预测算法,预测准确度比 Pentium III 提高了 33%;
- 4 096 个入口的分支目标缓存(Branch Target Buffer, BTB),为 Pentium III 的 8 倍。

(4) 新型的缓存体系结构

① L1 缓存包括高级执行跟踪缓存和数据缓存。

- 高级执行跟踪缓存用来存储已经过译码的指令;
- 执行跟踪缓存消除了主执行循环的译码延迟;
- 执行跟踪缓存把程序执行的控制流集成到一个单一的管线中;
- 低延迟的数据缓存读取延迟仅为 2 个时钟周期(主频为 2 GHz 时其为 1 ns),其容量为 8 KB,采用 4 路联合(4-Way Associative)方式工作,并使用 64 B 的缓存管道,其双端口结构能在一个时钟周期内完成读和写两种不同的操作。

② L2 缓存。

- CPU 内建的 L2 Cache 被称为高级传输缓存。其容量为 256 KB,与 Pentium III 的 L2 缓

存相同。采用8路联合(8-Way Associative)方式工作。L2缓存按 $2 \times 64\text{ B}$ 进行组织。当它与系统部件(内存、AGP或PCI总线等)传输信息时,都是以64 B为单位,确保了大量信息突发传输时的最佳性能。

• L2缓存的时钟速度与CPU核心相同,与CPU内核连接的专用总线宽度为256位,由此可以计算出L2缓存与CPU内核之间的传输带宽:主频为1.3 GHz的CPU为41.6 GB/s。目前最高主频为2 GHz的CPU则可达到64 GB/s。

(5) 与NetBurst微体系结构系统总线相连的高性能4倍速总线接口

Pentium III的系统总线频率为133 MHz,每时钟周期可传输64位的数据,即CPU的总线带宽为 $8\text{ bit} \times 133\text{ MHz} = 1.066\text{ GB/s}$ 。Pentium 4的系统总线频率仅为100 MHz,但由于每个时钟周期可传输4次数据,因此CPU总线带宽可高达 $8\text{ bit} \times 100\text{ MHz} \times 4 = 3.2\text{ GB/s}$ 。这项技术使得Pentium 4与其他部件(如内存)之间的数据传输率比目前所有的X86处理器都快得多,从而解决了系统的数据传输瓶颈。

但是,Pentium 4的高总线速率必须要有系统内其他部件的配合才能发挥其能力,所以Intel公司选择了Rambus DRAM(RDRAM),并配合以850芯片组,来构成Pentium 4系统的内存。这种内存结构能提供与Pentium 4系统总线相同的3.2 GB/s数据传输带宽。

- ① 超标量的并行执行机制;
- ② 1个扩充的硬件寄存器,通过重命名避免了寄存器名字空间的限制;
- ③ 128 B的缓存传输流水线,以64 B为单位进行传输;
- ④ 硬件指令预取。

2) NetBurst微体结构的组成

硬件预取部件可识别CPU内核执行程序的信息存取样本,并根据样本来“猜测”下次会被处理的信息,然后将这些信息预先读入高速缓存中。在处理大量数组型数据的应用中,Pentium 4的硬件预取可极大地加速CPU的执行效率。

图3.33所示为NetBurst微体系结构的框图。这种微体系结构包括3个主要组成部分:有序执行的前端(Front End)流水线、乱序推测执行的内核、有序指令流卸出部件。下面对这几个组成部分进行简要的介绍。

(1) 前端流水线

前端的作用是按照程序原来的顺序,为具有极高运行速度并能以 $1/2$ 个时钟周期的延迟执行基本整型数据运算的乱序执行内核提供指令。前端执行取指操作并对指令译码,然后把它们分解为简单的微操作。前端能在一个时钟周期内以程序原来的顺序向乱序执行内核发出多个微操作。

前端完成以下几个基本的功能:

- ① 预取那些可能要被执行的指令;

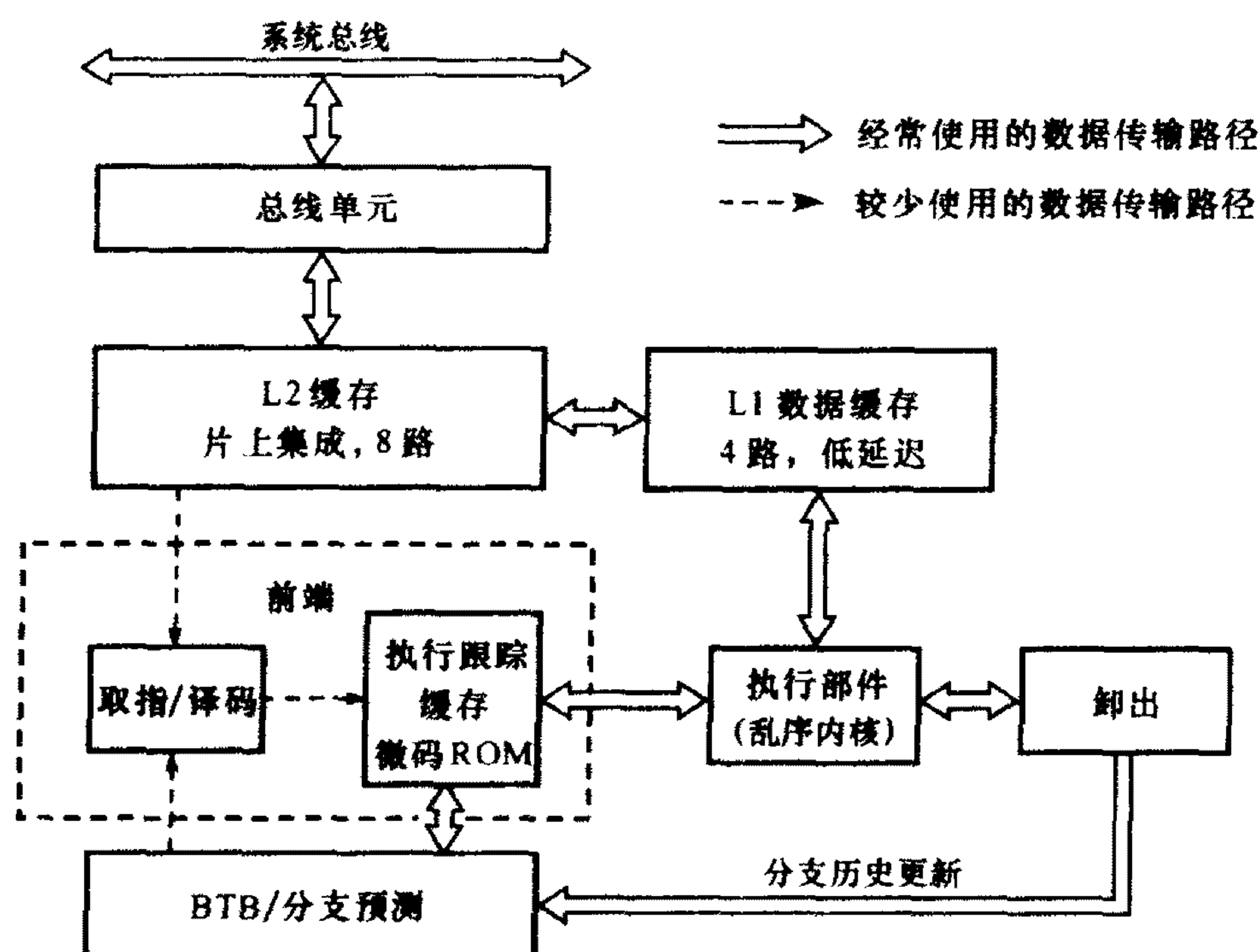


图 3.33 NetBurst 微体系结构

- ② 取出未被预取的指令；
- ③ 把指令译码成微操作；
- ④ 为复杂指令和特殊指令生成微码；
- ⑤ 从执行跟踪缓存送出译码后的指令；
- ⑥ 使用先进的预测算法来预测可能的程序分支。

Intel NetBurst 微体系结构的前端在设计时就考虑到了一些在高速流水线微处理器中常见的问题。其中对延迟影响最大的两个问题是：

- 指令译码时间；
- 由于分支或分支目标位于缓存流水线的中间，从而造成译码时间的浪费。

为了解决这两个问题，在 Pentium 4 中取消了 L1 指令缓存，而代之以执行跟踪缓存。把已译码的指令保存在执行跟踪缓存中。Intel 公司的设计人员认为，在原来的 P6 微体系结构中，L1 指令高速缓存中的指令直到真正要被处理单元执行时才会取出进行译码，这样，对某些复杂的 X86 指令需耗费太多的时间进行指令译码，以至于拖延整个流水线执行的运作。另外，在循环程序中，一段 X86 指令会被循环地多次执行，这样就使得每当这些指令进入执行路径一次就不得不再次进行译码。此外，当程序中的分支跳转预测错误时，也必须重新填充 L1 缓存，这是 L1 指令高速缓存难以处理的问题。使用了执行跟踪缓存后，当重复执行某些指令时，就可从执行跟踪缓存取出译码后的指令直接执行，从而节省了这些指令的译码时

间,避免了流水线的延迟。最重要的是,当超长流水线执行中出现分支预测错误时,流水线能及时从执行追踪缓存中快速地重新取得发生错误前已经过译码的指令,从而加速流水线填充过程。执行追踪缓存每两个时钟周期为流水线提供6个微指令,也就是每时钟周期提供3个微指令。追踪高速缓存大小为96 KB,一条 Pentium 4 的微指令长度约为64位,所以追踪高速缓存可容纳约12 000条微指令。

前端的执行过程是,首先由译码引擎取出指令并将其译码,然后经由微指令排序器(Micro Instruction Sequencer)将其序列化成一系列的微操作——称为轨迹(Traces)。这些微操作轨迹被存放在跟踪缓存中。一个分支指令所要转移到的可能性最大的目标轨迹紧跟在分支指令的轨迹后面,而不管实际的分支指令下面的一条指令是什么。一旦轨迹被建立,就在跟踪缓存中查找跟在这个轨迹后面的那条指令。如果该指令是已存在轨迹中的第一条指令,从存储器中取指并进行译码的操作就会停止,跟踪缓存就成为下一条指令的来源地。NetBurst 微体系结构中关键的执行循环如图 3.33 所示,它比图 3.32 所示的 P6 微体系结构中的关键执行循环要简单。

(2) 乱序执行内核

乱序执行能力是并行处理的关键所在。乱序执行使得处理器能够重新对指令排序,这样,当一个微操作由于等待数据或竞争执行资源而被延迟时,后面的其他微操作也仍然可以绕过它继续执行。处理器拥有若干个缓冲区来平滑微操作流。这意味着当流水线的部分产生了延迟,该延迟也能够通过其他并行的操作予以克服,或通过执行已进入缓冲区中排队微操作来克服。

乱序执行内核按并行执行的要求来进行设计。它能在一个周期中发出6个微操作,这大大超过跟踪缓存和卸出部件执行微操作的速率。大多数流水线能够在每一个周期启动执行一个新的微操作,所以每条流水线允许一次穿越多条指令。

(3) 指令卸出

卸出部分接收执行核心的微操作执行结果并处理它们,以便根据原始的程序顺序来更新相应的程序执行状态。为了保证执行在语义上正确,指令的执行结果在卸出前必须按照原始程序的顺序进行提交。

当一个微操作执行完成,并把结果写入目标后,它就被卸出。每一周期被卸出的微操作多达3个。处理器中的重排序缓冲器(Re-Order Buffer, ROB)就是实现此功能的部件。它缓冲执行结束的微操作、按原始顺序更新执行状态以及管理异常的排序。

卸出部分还跟踪分支的执行,并把更新了的转移目标送到 BTB,以更新分支历史。这样,不再需要的轨迹被清除出跟踪缓存,并根据更新过的分支历史信息来取出新的分支路径。

5. Pentium 4 系统结构

自从 80386 推出以来到现在最新推出的 Pentium 4, Intel 公司的 CPU 体系结构基本没有

什么大的变化, Intel 公司将这一类 CPU 的体系结构通称为 IA-32 结构。图 3.34 对 IA-32 结构中的系统寄存器和数据结构进行了简要总结。

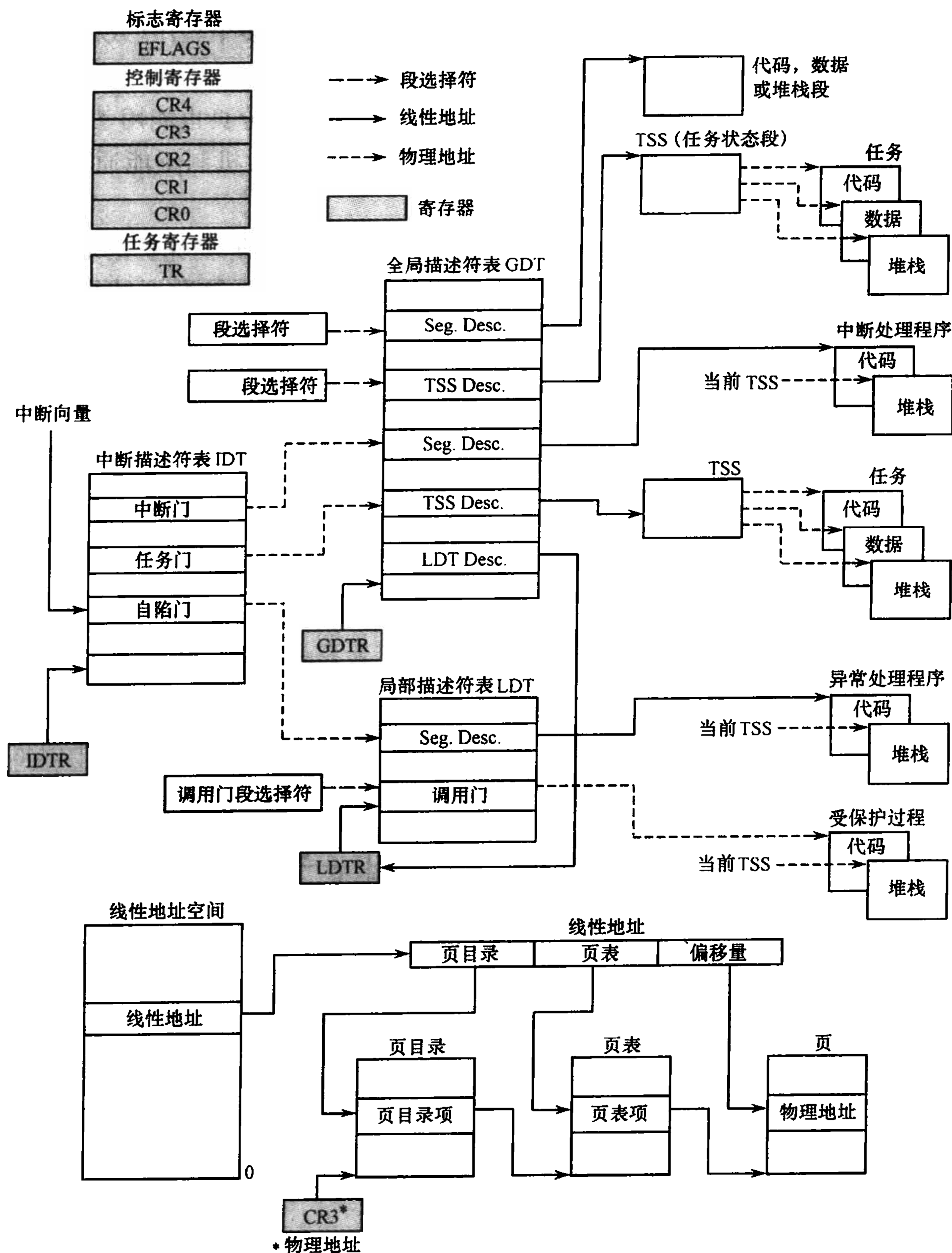


图 3.34 IA-32 结构的系统寄存器和数据结构

6. Pentium 4 的存储管理

Pentium 4 继承了 IA-32 结构,所以它的存储器管理与 80386 基本相同,也包括了分段管理和分页管理。分段提供了隔离代码、数据和堆栈的机制,使多个程序(或任务)能够运行在同一个 CPU 上,而不会互相干扰。分页提供了实现传统的基于页请求的虚存系统。这种系统当需要时能把程序执行环境的片段映射到物理存储器中。分页也能用于多个任务的隔离。当运行在保护方式时,必须使用某种形式的分段机制。分段机制不能通过状态位被禁止掉,而分页则是可选的。

把处理器的可寻址存储空间(称为线性地址空间)分成较小的、受保护的地址空间,称为段,如图 3.35 所示。段可用来存放代码、数据和堆栈,或者存放系统数据结构(例如 TSS 或 LDT)。当多个程序运行在同一个处理器上时,每一个程序都能够指定自己的段集合。处理器将限制这些段的界限,以保证这个程序不会把数据写到其他程序的段中,干扰其他程序的运行。分段机制也允许指定段类型,以限制对某些特殊段的操作。

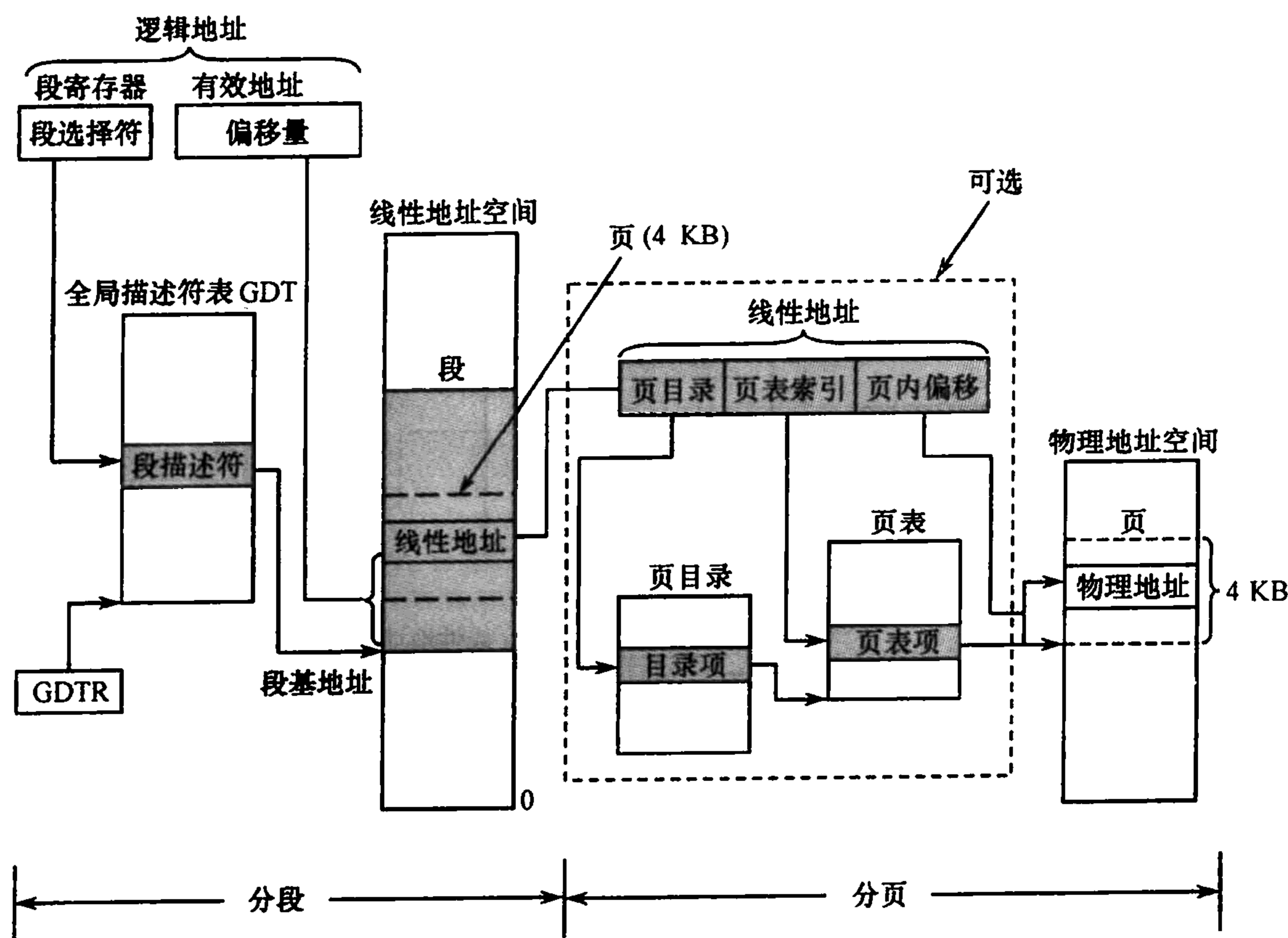


图 3.35 保护方式下存储器管理中的分段与分页

所有段都包含在处理器的线性地址空间中。为了定位某个段中的一个字节,则必须提供该字节的逻辑地址(又称为远指针)。正如已经知道的,逻辑地址是由段选择符和偏移量

两部分构成。段选择符是段的惟一标识。它提供了访问段描述符表(如 GDT)的偏移地址(或索引)。段描述符表中存放的是称为段描述符的数据结构。每一个段都有一个段描述符,段描述符定义了段的大小、访问权限和特权级、段的类型以及该段第一个字节在线性地址空间中的位置(称为段的基地址)。逻辑地址中的偏移量与段基地址相加就可以定位段中的任意字节。段基地址加偏移量得到的值称为线性地址。若未使用分页,处理器的线性地址空间直接映射为物理地址空间。物理地址空间定义为处理器在它的地址总线上所能产生的地址范围。

由于多任务系统通常定义了一个比其拥有的物理存储器大得多的地址空间,这就需要有一个将线性地址空间虚拟化的方法。线性地址空间虚拟化是通过分页来实现的。

分页支持虚拟存储器环境,在这种环境中,用一个小容量的存储器(RAM 和 ROM)和一些磁盘空间来模拟一个非常大的线性地址空间。当使用分页时,每一个段都分为多个页(页面大小通常为 4 KB),页可存储在内存中或磁盘中。操作系统负责维护页目录和一个页表集合,以跟踪页的使用。当一个程序(或任务)要访问线性地址空间中的一个地址位置时,处理器使用页目录和页表把线性地址转换为存储器的物理地址,然后即可执行所请求的动作(读或写)。如果所访问的页不在物理存储器中,处理器就会暂时中断该程序的执行(通过产生一个页错误异常),由操作系统把所需的页从磁盘读入物理内存中,然后接着执行由于页错误而被中断的程序。在物理内存和磁盘之间的页交换对应用程序来说是透明的。当处理器运行在虚拟 8086 模式时,为 16 位 CPU 编写的程序也可以分页。

7. Pentium 4 的基本执行环境

Pentium 4 仍然继续支持 IA-32 结构的三种操作模式:保护模式、实模式和系统管理模式。Pentium 4 的基本执行环境对这三种模式来说都是相同的。Pentium 4 基本执行环境包括以下可使用的资源(如图 3.36 所示):

① 地址空间 任何程序或任务都可以访问最大为 4 GB(2^{32} B)的线性地址空间和最大为 64 GB(2^{36} B)的物理地址空间。

② 基本的程序执行寄存器 包括 8 个通用寄存器、6 个段寄存器、1 个标志寄存器和 1 个指令指针寄存器。这些寄存器能够以字节、字和双字来执行基本的整型算术逻辑运算,进行程序流控制,进行位和字符串的操作,以及访问存储器等。

③ 80X87 FPU 寄存器 包括 8 个 80X87 FPU 数据寄存器、80X87 FPU 控制寄存器、80X87 FPU 状态寄存器、80X87 FPU 指令指针寄存器、80X87 FPU 操作数指针寄存器、80X87 FPU 标签寄存器和 80X87 FPU 操作码寄存器。这些寄存器用于单精度浮点数、双精度浮点数、扩充的双精度浮点数、字或双字或四字整型数、BCD 数的运算。

④ MMX 寄存器 8 个 MMX 寄存器用于执行单指令多数据(SIMD)操作,支持 64 位紧缩的字节、字和双字整数类型。

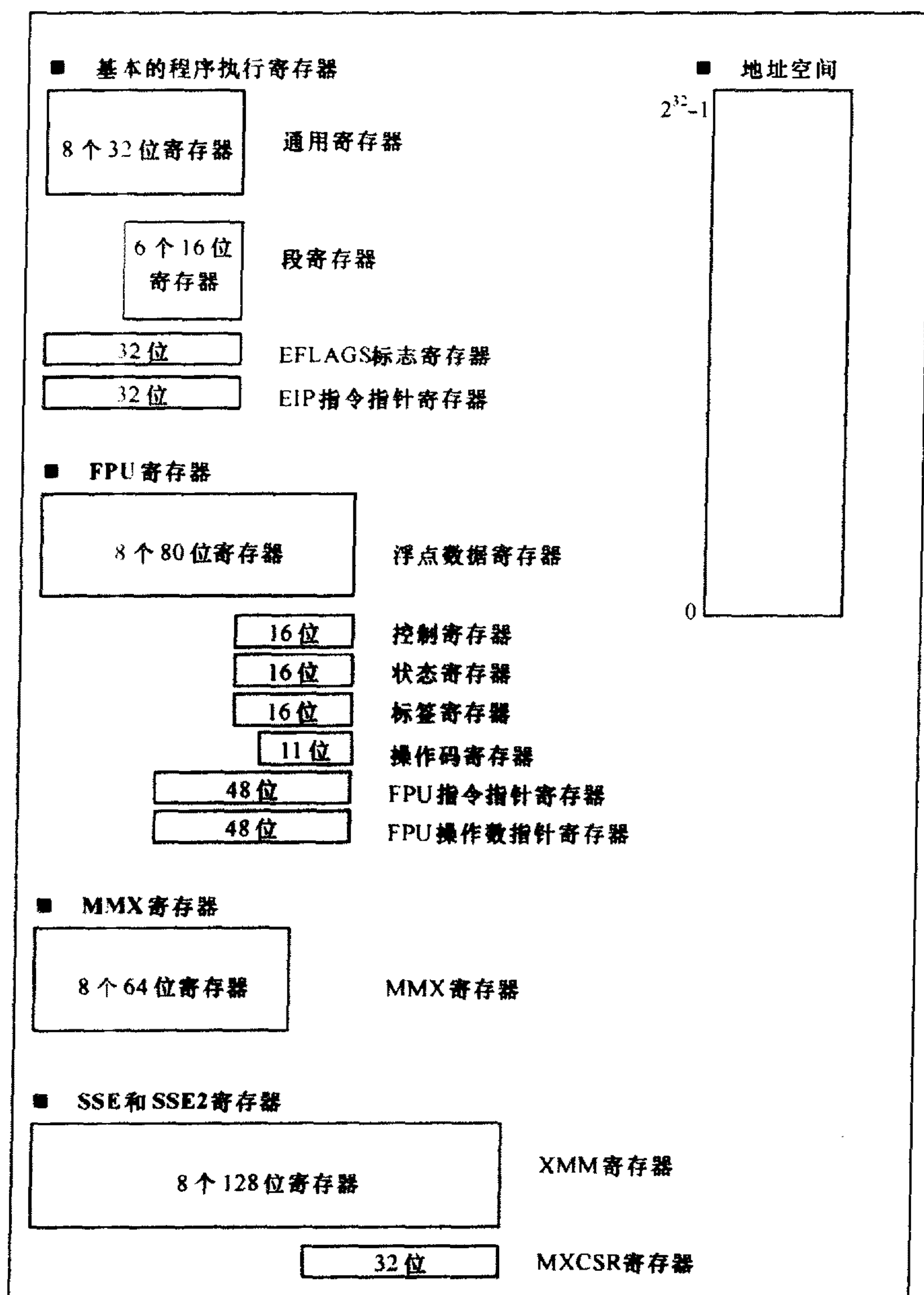


图 3.36 Pentium 4 的基本程序执行环境

⑤ XMM 寄存器 8 个 XMM 寄存器和 1 个 MXCSR 寄存器支持 128 位紧缩的单精度浮点数、双精度浮点数以及 128 位紧缩的字节、字、双字、四字整型数的 SIMD 操作。

⑥ * 堆栈 用于支持过程(子程序)调用和向过程传递参数。

- ⑦ * I/O 端口 用于实现与外部设备的连接。
- ⑧ 控制寄存器 5 个控制寄存器($CR_0 \sim CR_4$)决定了处理器的操作模式和当前任务的特征。
- ⑨ * 存储管理寄存器 GDTR、IDTR、任务寄存器和 LDTR 指出了保护模式下存储器管理所使用的数据结构在内存中的位置。
- ⑩ * 调试寄存器 $DR_0 \sim DR_7$ 用来控制和监视处理器的调试操作。
- ⑪ * 机器检测寄存器 用于检测和报告硬件错误。
- ⑫ * 存储器类型范围寄存器(MTRRs) 用于为物理存储器的范围指定存储器类型。
- ⑬ * 机器相关寄存器(MSRs) 用于控制和报告处理器的性能,它们不能被应用程序所访问(除了时间戳计数器外)。
- ⑭ * 性能监视寄存器 用于监视处理器性能事件。

在 Pentium 4 的寄存器中,通用寄存器、段寄存器、指令指针寄存器与 80386 完全相同,标志寄存器也只是增加了几个状态位。下面介绍标志寄存器中增加的几个状态位。至于 FPU 寄存器、MMX 寄存器和 XMM 寄存器,请参考有关文献。

图 3.37 给出了 Pentium 4 标志寄存器的各个标志位的情况。可以看出,Pentium 4 的标志位比 80386 增加了 4 位: AC、VIF、VIP 和 ID。

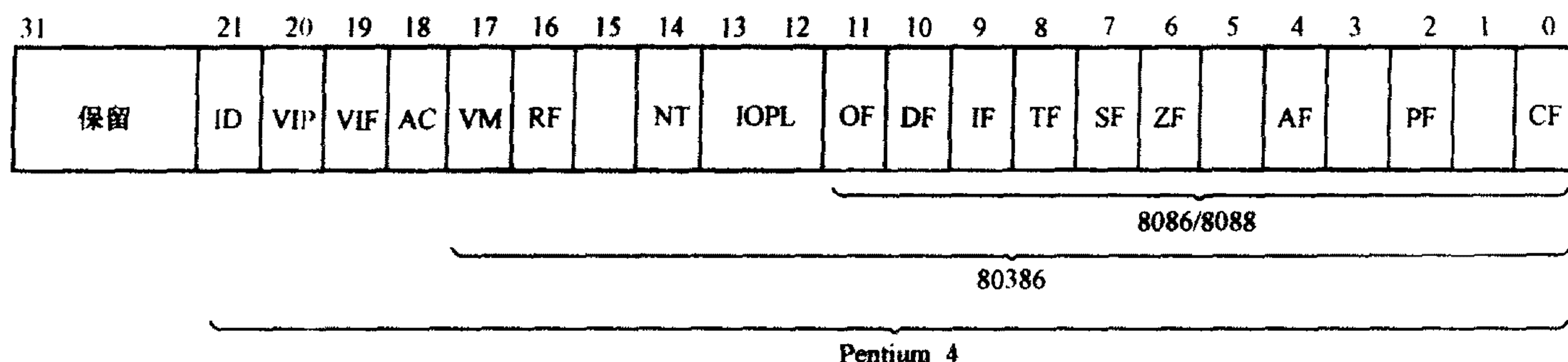


图 3.37 Pentium 4 的标志寄存器 EFALGS

AC(对齐检查标志) 当 $AC = 1$ 并且 CR_0 寄存器的 $AM = 1$ 时,允许存储器对齐检查。

VIF(虚拟中断标志) IF 标志位的虚拟映像。与 VIP 联合使用。

VIP(未决虚拟中断标志) $VIP = 1$ 时表示有未决的中断, $VIP = 0$ 表示没有未决的中断。

ID(鉴别标志) 如果程序能设置或清除这一位,表示可以使用 CUID 指令。

习 题 三

- 3.1 8086 微处理器与前一代微处理器相比有什么特点?
- 3.2 说明 8086 的 EU 和 BIU 的主要功能。在执行程序过程中它们是如何相互配合工作的?
- 3.3 在执行指令期间,EU 能直接访问存储器吗?为什么?
- 3.4 8086 的最小工作模式和最大工作模式有什么区别?
- 3.5 8086 的地址总线和数据总线的宽度分别是多少?什么因素决定 8086 能够寻址 1 MB 的存储器空间?
- 3.6 总线周期中,什么情况下要插入 T_w 等待周期?插入 T_w 周期的个数取决于什么因素?
- 3.7 在 8086 中标志寄存器包含哪些标志位?各位为 0(为 1)分别表示什么含义?
- 3.8 试说明 8086 内部寄存器组包括哪些寄存器?各寄存器的作用是什么?
- 3.9 8086 系统中,存储器为什么要分段?一个段最大为多少字节?最小为多少字节?
- 3.10 若已知某存储单元的逻辑地址为 1E00:3800H,其对应的物理地址是多少?
- 3.11 若 CS = 6000H,则当前代码段可寻址的存储空间的范围是多少?
- 3.12 8086 在最小模式下构成计算机系统至少应包括哪几个基本部分(器件)?若为最大模式呢?
- 3.13 在 8086 构成的微型计算机系统中,内存储器可分为几类逻辑段?程序代码存放在哪个逻辑段?
- 3.14 80386 包含哪些寄存器?各有什么主要用途?
- 3.15 什么是实地址模式?什么是保护模式?
- 3.16 80386 访问存储器有哪两种方式?分别可寻址多大的地址空间?
- 3.17 如果 GDT 寄存器值为 0013000000FFH,装入 LDTR 的选择符为 0040H,试问装入缓存 LDT 描述符的起始地址是多少?
- 3.18 页转换产生的线性地址的三部分各是什么?
- 3.19 一个描述符中有几个字节?试说明其中每一个域名及大小。
- 3.20 选择符 022416H 装入了数据段寄存器,该值指向局部描述符表中从地址 00100220H 开始的段描述符。如果该描述符的内容为:
 $(00100220H) = 1000H, (00100221H) = 2200H$
 $(00100222H) = 0000H, (00100223H) = 0010H$
 $(00100224H) = 1C00H, (00100225H) = 8000H$
 $(00100226H) = 0001H, (00100227H) = 0100H$
则段基地址和段界限各为多少?
- 2.21 Pentium 4 的基本程序执行环境包含了哪些寄存器?

第4章 总线结构

本章主要介绍总线的基本概念、不同分类方法及总线的结构层次和工作原理,另外,还系统地介绍几种常见系统总线及外部总线的特点、结构、工作原理、性能指标以及应用等。

4.1 总线的基本概念

4.1.1 概述

微型计算机从其诞生以来就采用了总线结构。CPU 通过总线实现读取指令,并实现与内存、外设之间的数据交换,在 CPU、内存与外设确定的情况下,总线速度是制约计算机整体性能的关键,先进的总线技术对于解决系统瓶颈、提高整个微型计算机系统的性能有着十分重要的影响,因此在微型计算机 30 多年的发展过程中,总线结构也不断地发展变化,总线结构方式已经成为微型计算机性能的重要指标之一。

从物理来看,总线(BUS)是一组传输公共信息的信号线的集合,是在计算机系统各部件之间传输地址、数据、控制和状态信息的公共通路。它由一组导线和相关的控制、驱动电路组成。在处理器内部的各功能部件之间,在处理器与高速缓冲存储器 and 主存之间,在处理器系统与外围设备之间以及网络系统的各节点之间等等,都是通过总线连接在一起的。目前在微型计算机系统中常把总线作为一个独立的功能部件来看待。

总线是以分时的方法来为多个部件服务的。总线的工作方式通常是由发送信息的部件分时地将信息发往总线,再由总线将这些信息同时发往各个接收信息的部件。究竟哪个部件接收信息,要由 CPU 给出的设备地址经译码产生的控制信号来决定。

对于同时出现的数据传输请求,总线应根据某种策略来决定首先为谁服务,这个工作是由总线仲裁电路来实现的。总线仲裁电路可根据设备的优先级别对总线请求进行排队,使发出请求的设备按其优先级的高低依次使用总线,以避免总线冲突。

对最终用户来说,总线的指标主要有 2 个,总线的工作频率和总线的宽度。总线频率是指协调总线工作的“心跳”信号——总线时钟频率,总线频率越高,单位时间内传输的信息就越多。总线的宽度是指能够一次并行传送的信息位数,同样,总线的宽度越宽,单位时间内传输的信息也就越多。所以,总线的数据传输速率正比于总线的频率和总线的宽度。

在微型计算机系统中除了采用总线技术外,还采用了标准接口技术。其目的也是为了便于模块结构设计,可以得到多个厂商的广泛支持,便于生产与之兼容的外部设备和软件。接口一般是指主板和某类外设之间的适配电路,其功能是解决主板和外设之间在电压等级、信号形式和速度上的匹配问题。因此不同类型的外设需要不同的接口,不同的接口是不通用的。例如,硬盘和软盘驱动器的接口是不兼容的,因此不能在硬盘接口上接入软盘驱动器。另一方面,由于目前的一些新型接口标准,如 USB、IEEE 1394 等,允许同时连接多种不同的外设,因此也把它们称为外设总线。此外,连接显示系统的接口 AGP,由于习惯上的原因(原来的显示卡要插入 ISA 或者 PCI 总线插槽中),也被称为 AGP 总线,但是实际上它应该是一种接口标准。

4.1.2 总线的分类

1) 总线按相对于 CPU 与其他芯片的位置可分为片内总线和片外总线

在 CPU 内部,各个主要部件之间(例如寄存器之间,ALU 与控制部件之间)也是用总线连接起来的,这个 CPU 内部的总线就称为片内总线(即芯片内部的总线);而通常所说的总线(Bus)则是指片外总线,是 CPU 与内存和输入/输出设备接口之间进行通信的通路。有的资料上也把片内总线叫做内部总线或内总线(Internal Bus),把片外总线叫做外部总线或外总线(External Bus)。

2) 按总线传送信息的类别,可把总线分为地址总线、数据总线和控制总线

通常所说的总线都包括这三个组成部分。地址总线(Address Bus, AB)用于传送存储器地址码或输入/输出设备地址码,数据总线(Data Bus, DB)用于传送指令或数据,控制总线(Control Bus, CB)用来传送各种控制信号。例如,ISA 总线共有 98 条引脚(或称 ISA 插槽有 98 条引脚),其中数据线有 16 条(构成数据总线),地址线 24 条(构成地址总线),其余为控制信号线(构成了控制总线)、电源线和地线。

3) 按照总线传送信息的方向,可把总线分为单向总线和双向总线

单向总线的功能是使挂在总线上的一些部件将信息有选择地传向另一些部件,而不能反向传送。双向总线则能使任何挂在总线上的部件或设备之间互相传送信息。地址总线属于单向总线,方向是从 CPU 或其他总线主控设备发往存储器或设备接口。数据总线属于双向总线,即可以从 CPU 送出,也可以从外部送入 CPU。控制总线属于混合型总线,虽然控制总线中的每一根控制线方向是单向的,但各控制线的方向相对于 CPU 来说却有输入,有输出,所以从整体上通常也把它称为双向总线。

4) 按总线的层次结构可分为 CPU 总线、存储总线、系统总线和外部总线

CPU 总线用来连接外部控制芯片。存储总线专指连接到存储控制器和 DRAM 的总线。有些 CPU 的体系结构中,把 CPU 总线和存储总线合并起来,统称为前端总线(Front Side Bus,

FSB)。

系统总线也称为 I/O 通道总线,用来与 I/O 扩充插槽上的各扩充接口卡相连接。系统总线有多种标准,以适用于各种系统。

外部总线用来连接外设控制芯片,如主板上的 I/O 控制器和键盘控制器。

5) 按总线在微型计算机系统的位置可分为机内总线和外设总线

上面介绍的各类总线都属于机内总线。外设总线(Peripheral Bus)是指用来连接外部设备的总线,实际上是一种外设的接口。目前在微型计算机上流行的接口标准有 IDE(EIDE)、SCSI、USB 和 IEEE 1394 四种。前两种主要用来连接硬盘、光驱等存储设备,后面两种新型外部总线可以用来连接各种外部设备。

6) 系统总线

微型计算机上的系统总线又可分为 ISA、EISA、MCA、VESA、PCI、AGP 等多种标准。

4.2 总线结构的类型

4.2.1 总线的系统结构

总线技术的发展与微型计算机系统结构的发展密不可分,在很大程度上,微型计算机系统结构决定了其所采用的总线结构。从微型计算机系统结构的观点来考察,总线结构大致分为单总线结构和双/多总线结构。

1. 单总线结构

单总线结构如图 4.1 所示。系统的各个部件均接在单总线上,构成微型计算机的硬件系统,所以它又称为面向系统的单总线结构。

在早期的微型计算机中,单总线结构应用非常广泛,例如 APPLE 微型计算机、IBM PC、PC/XT 及其兼容的微型计算机系统均采用了单总线结构。

在单总线结构中,CPU 与主存之间、CPU 与 I/O 设备之间、I/O 设备与主存之间、各种设备之间都通过单一的总线交换信息。具有单总线结构的微型计算机系统其优点是体系结构简单,软硬件开发方便,用户易于扩充系统 I/O 设备,而且在主存与 I/O 设备交换信息时还允许 CPU 继续工作。但由于微型计算机系统的所有设备部件均挂在单总线上,所以这种结构就限制了单总线只能分时工作,即同一时刻只能在两台设备之间传送数据,这就使得系统总体数据传输的效率和速度受到限制,这是单总线结构的主要缺点。另外,这种结构将存储器与 I/O 接口同等看待,在设计总线时就不得不考虑兼顾双方的特点,使得 CPU 访问存储

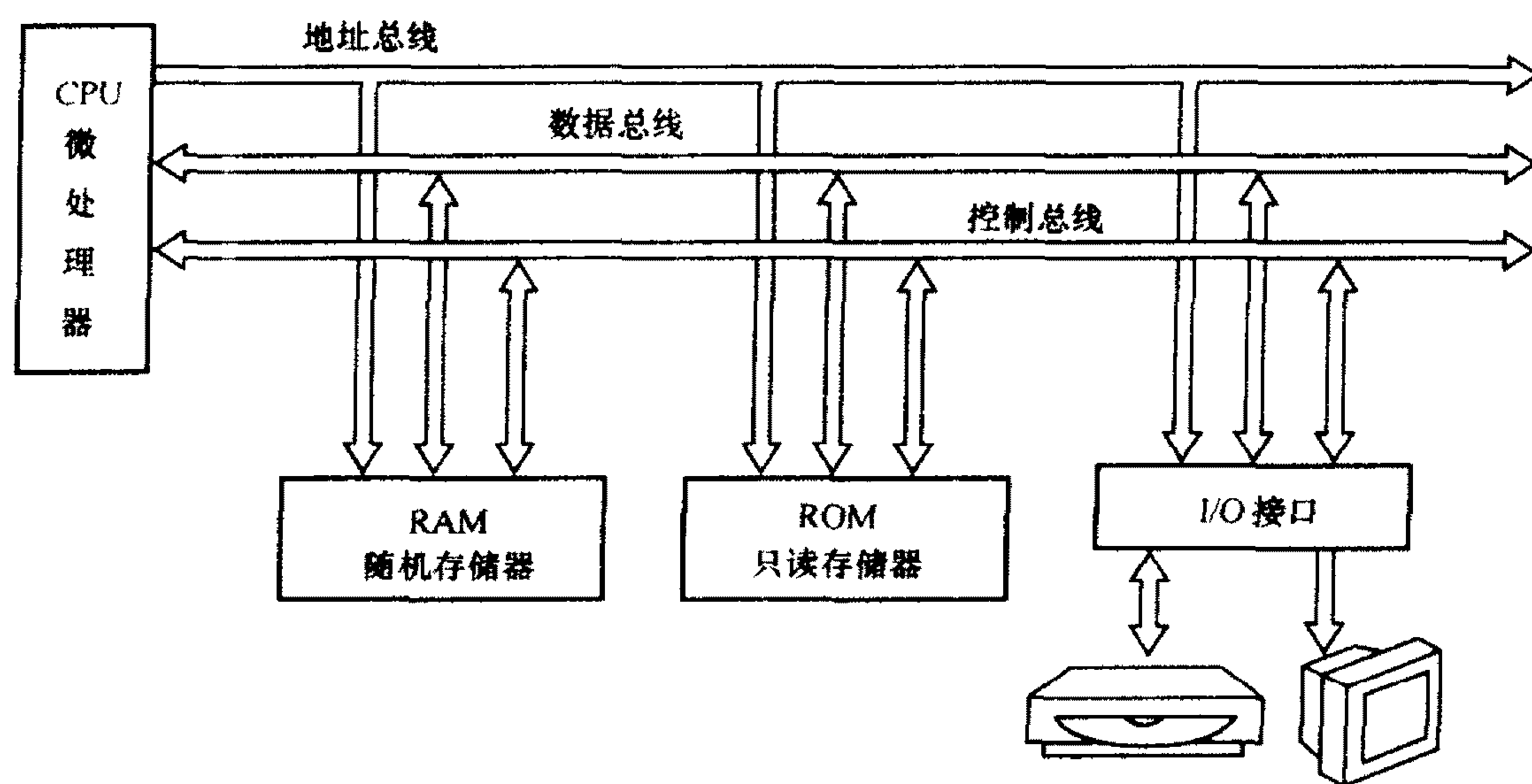


图 4.1 单总线结构

器的速度也受到了很大的影响。

2. 面向 CPU 的双总线结构

解决 CPU 与存储器之间数据传送速度的方法之一就是设置另一条连接 CPU 与存储器的专用总线。这就是面向 CPU 的双总线结构,如图 4.2 所示。双总线结构中有两组总线,一组总线是 CPU 与主存储器之间进行信息交换的公共通路,称为存储器总线。CPU 利用存储总线从主存储器取出指令后进行分析、执行,再将结果送回主存储器。另一组是 CPU 与 I/O 设备之间进行信息交换的公共通路,称为输入/输出(I/O)总线。各外围设备通过接口电路挂接在 I/O 总线上,接口是主机与外部设备之间的数据交换部件,它一般由缓冲寄存器及有关控制逻辑组成。

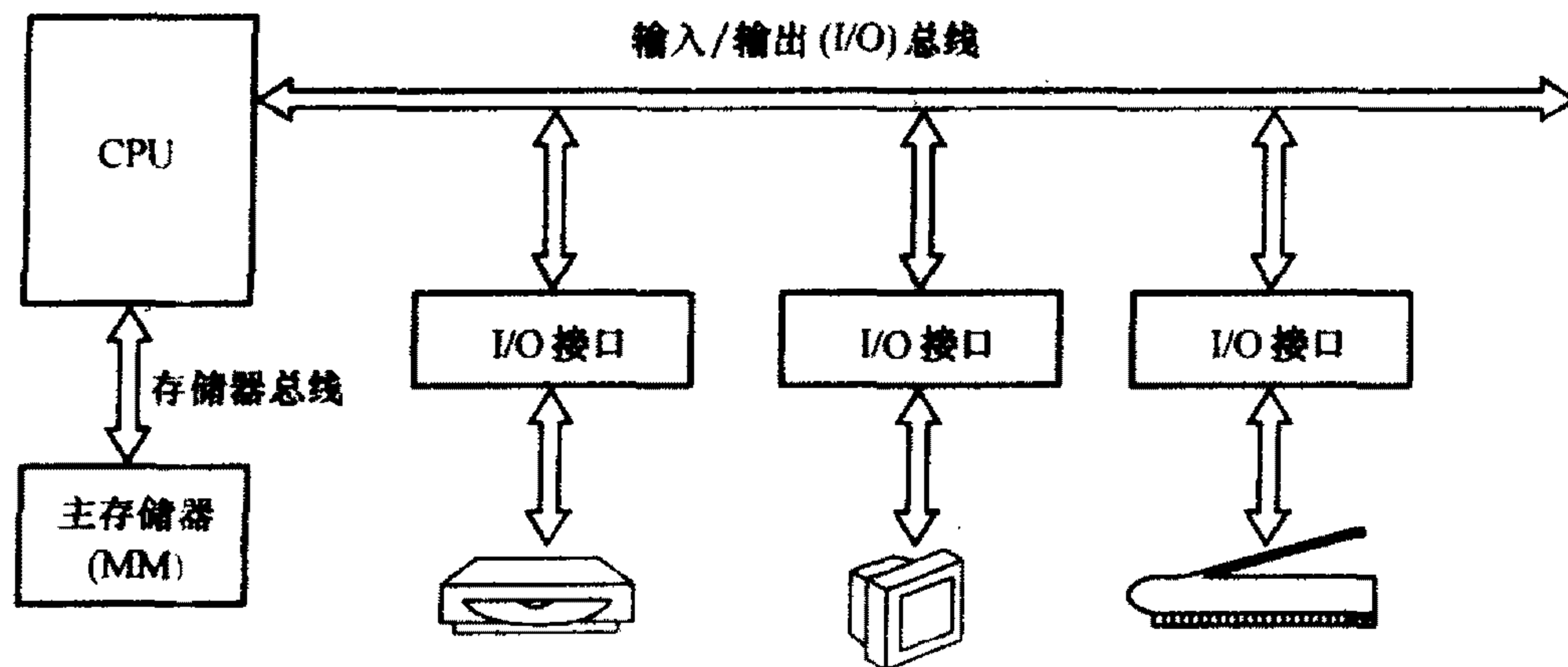


图 4.2 面向 CPU 的双总线结构

由于在 CPU 与主存储器之间、CPU 与 I/O 设备之间分别设置了总线,使得二者之间的总线冲突现象大大减少,而且可根据二者的特点分别设计总线,从而提高了微型计算机系统整体的信息传送速率和传送效率。但是,由于外围设备与主存储器之间没有直接的通路,它们必须通过 CPU 才能进行信息交换。也就是说当输入设备与主存储器进行信息交换时,必须要通过 CPU 来转发。例如,输入设备要把信息送往主存储器时,首先要送到 CPU 的某个寄存器中,然后再将该寄存器中的信息送入主存;当输出运算结果时,也必须先由主存储器送入 CPU 的寄存器中,然后再送到某一指定的输出设备。这势必增加 CPU 的负担,CPU 必须花大量的时间进行信息的输入/输出处理,从而降低了 CPU 的工作效率,或称为 CPU 占用率高。一般来说,外设工作时要求 CPU 干预越少越好。CPU 干预越少,这个设备的 CPU 占用率就越低,说明设备的智能化程度越高。CPU 占用率与系统结构有很大关系。

3. 面向主存储器的双总线结构

如图 4.3 所示,面向主存储器的双总线结构保留了单总线结构的优点,即所有设备和部件均可通过总线交换信息。与单总线结构不同的是在 CPU 与主存储器之间,又专门设置了一组高速总线,使 CPU 可以通过它直接与主存储器交换信息。面向主存储器的双总线结构不仅使信息传送效率提高,而且减轻了总线的负担,这是它的主要优点;但这种总线结构控制电路比较复杂,硬件造价较高。高档微型计算机中通常采用这种面向存储器的双总线结构。

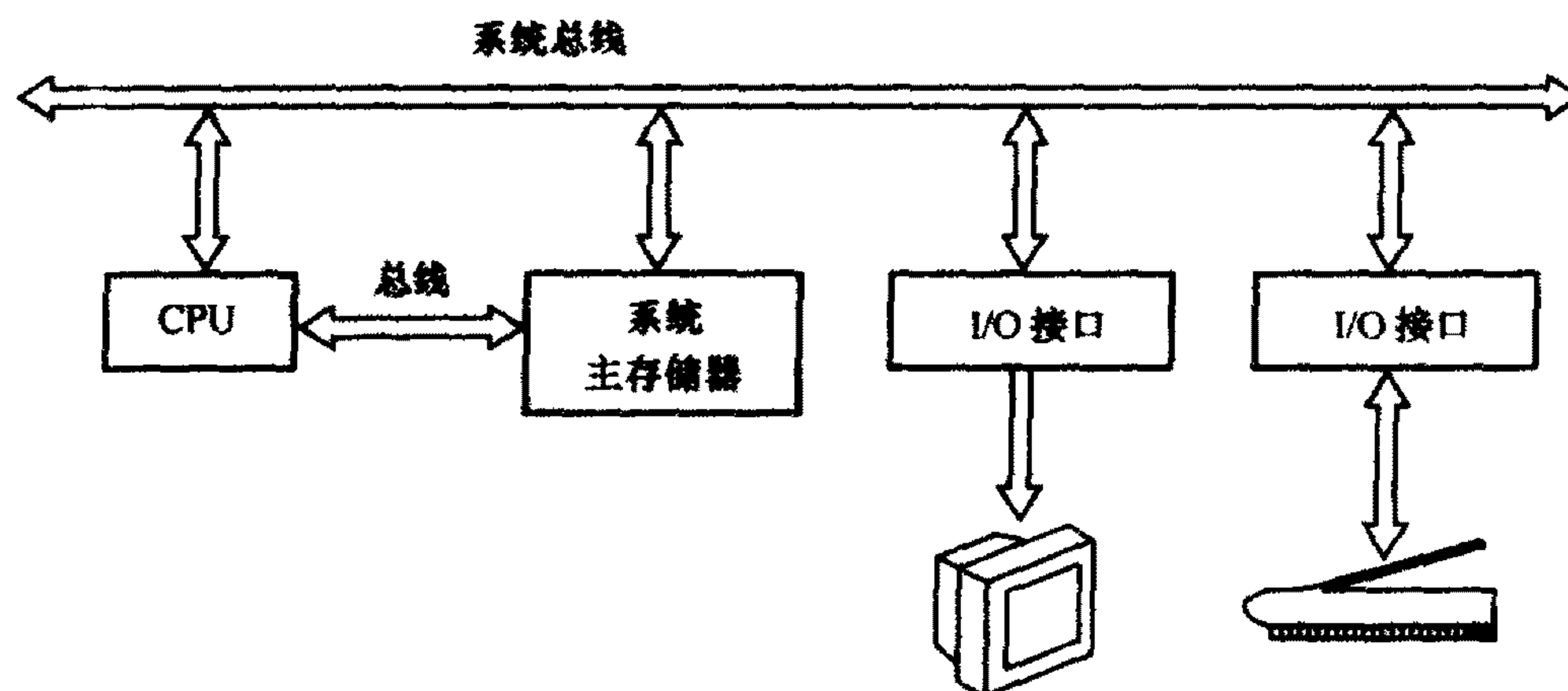


图 4.3 面向主存储器的双总线结构

4. 多总线结构

除以上的单总线结构和双总线结构外,现在的微型计算机系统中常见的还有多总线结构,即系统中拥有两个以上的总线。例如,Pentium III 和 Pentium 4 系统就有 3 条以上的总线:前端总线、PCI、ISA、AGP 以及 USB 总线等,如图 4.4 所示。

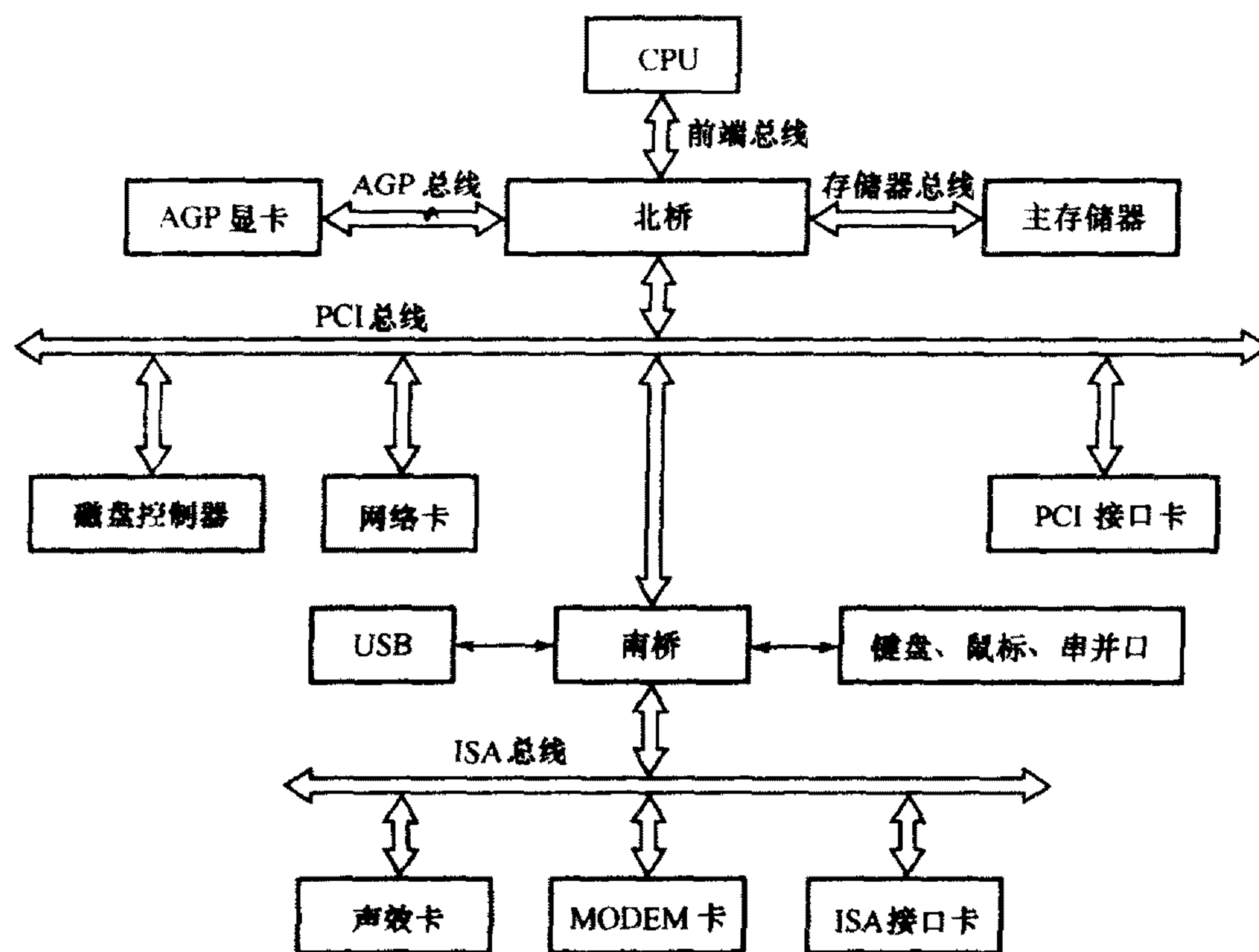


图 4.4 多总线结构

多总线结构将不同类型的设备划分到不同的总线层次中,这样做的目的是兼顾速度和成本的要求。速度越快的部件,它所连接的总线拥有的传输带宽就越高,但其生产成本很高,价格很贵。反之,用于连接速度较低部件的总线对带宽的要求较低,这样相关电路的开发和制造就较为简单,从而降低了生产成本。

在 Pentium 4 系统中,前端总线因为与 CPU 直接相连,其带宽可高达 4.2 GB/s, AGP 总线的带宽为 1 GB/s,存储器总线的带宽为 2.1 GB/s。相比之下,PCI 总线的带宽为 133 MB/s, USB2.0 总线的带宽为 60 MB/s(480 Mb/s),而 ISA 总线的带宽仅为 8 MB/s。

4.2.2 总线的层次结构

计算机的总线系统是由处于计算机系统不同层次上的若干个总线组成的,一般可分以下几个层次的总线类型: CPU 总线、系统总线、局部总线、外部总线等。

1. CPU 总线

系统主板内含有 CPU、ROM、RAM、控制芯片组和 I/O 接口芯片,这些芯片之间有许多信号连接关系,其中包括地址、数据和控制信息,它们都要通过 CPU 总线来传输。CPU 总线作为 CPU 与外界的公共通道实现了 CPU 与主存储器、CPU 与 I/O 接口和多个 CPU 之间的连

接,并提供了与系统总线的接口。CPU 总线一般是生产厂家针对其具体的处理器而设计的,是与处理器相关的,无法实现标准化,因此尚没有统一的规范。

CPU 总线包括地址总线(CAB),数据总线(CDB)和控制总线(CCB)三部分。其中地址总线是单方向的(输出),数据总线是双向的(输入/输出),控制总线情况比较复杂,它包括很多具体的控制线,每一根控制线不是输出就是输入(不可能有哪一根控制线像数据总线那样是双向的)。所以从总体上来看,控制总线是双向的,但具体到某个控制线就是单向的。

图 4.5 所示的是 80386 的总线信号,图中除数据总线和地址总线外,其他均属于控制总线的范围,控制总线每个信号线的含义请参考有关资料。

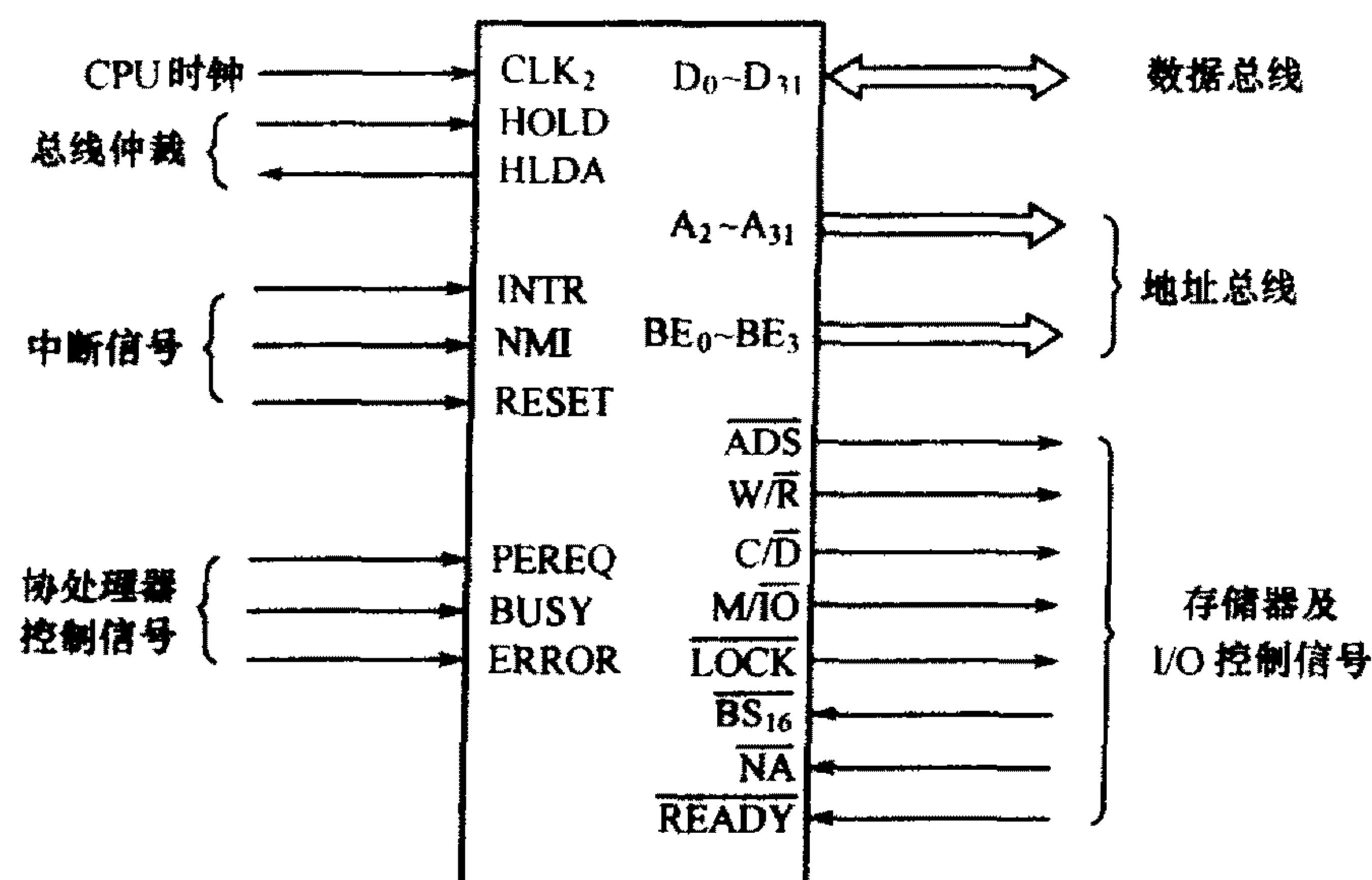


图 4.5 80386 的总线信号

80386 是高性能的微处理机,它通过高性能总线与存储器和外围设备连接,为使存储器接口和 I/O 接口尽可能简单,80386 的总线信号经过慎重的选择,使用了分开的地址与数据引脚,以支持高速的总线数据传输及总线流水线操作。它还使用一些引脚作为周期定义、周期控制、中断、总线仲裁与协处理器支持等。80386 局部总线在系统内是同步 32 位多路分配的数据和地址总线。在 16 MHz 时钟频率下,其传送速率为 32 MB/s。

数据总线是由 D₃₁ ~ D₀ 这 32 根三态数据线构成的。即可按 16 位传送,也可按 32 位传送。具体按多少位传送,由 \overline{BS}_{16} 信号控制, $\overline{BS}_{16} = 0$ 时,为 16 位传送; $\overline{BS}_{16} = 1$ 时,为 32 位传送。

地址总线是由 A₃₁ ~ A₂ 30 根地址线和 BE₀ ~ BE₃ 4 根字节选通线构成。每根字节选通线对应于一次传送的 4 个字节(共 32 位)中的一个。因为 80386 一次可传送可能是 1 个字节、2 个字节、3 个字节或 4 个字节,所以必须使用字节选通线指明 32 位数据线上的数据中哪些位

(或4B中哪些字节)是有效的。指令在存取数据时是以字节、字或者双字三种方式进行的。若存取一个字或双字时没有对准边界,80386就自动地产生两个总线周期,以便完成这次存取。

在80386总线设计中,主要考虑的是大量的、各种各样的外部设备与各类存储器之间的适应和配合问题。为简化总线的接口逻辑,仅用一个信号 \overline{ADS} 表示新总线周期的开始和地址输出是有效的。而 W/\overline{R} 、 C/\overline{D} 、 M/\overline{IO} 等总线周期定义信号表示执行总线周期的类型。例如,当前执行的总线周期是读还是写,是数据传送还是总线控制,是访问存储器空间还是I/O空间等周期类型。

总线仲裁信号HOLD和HLDA准许其他设备获得对总线的控制。这样就允许如Intel的数学协处理器、DMA控制器、82586 LAN控制器等设备与80386并行操作。还可以用这个信号隔离80386与总线的联系,以达到测试的目的。

80386还要调节速度比较慢的存储器和外围子系统间的关系。 \overline{READY} 信号用于表示总线存取已完成,外部子系统亦可保持 \overline{READY} 有效,以使80386用连续插入等待状态的办法来展宽当前的总线周期,直到子系统完成这次存取为止。

80386使用总线周期来完成对存储器和I/O端口的读/写操作以及中断响应。总线周期与三组信号有关: $A_{31} \sim A_2$ 、 $BE_0 \sim BE_3$ 地址信号; W/\overline{R} 、 C/\overline{D} 、 M/\overline{IO} 周期定义信号,决定总线周期的操作类型和操作对象; \overline{ADS} 地址状态信号,用于指明一个总线周期的开始。

当地址在总线上有效、周期定义信号指明了对应的总线周期类型、且 \overline{ADS} 为低电平时,一个总线周期就开始了。

80386中,一个 CLK_2 时钟周期称为一个总线状态,最快的80386总线周期需要2个总线状态。所以对存储器或I/O接口的访问最快也要2个总线状态。

80386的总线周期按照CPU的工作状态可分为读/写总线周期、中断响应周期、暂停和停机周期。读/写总线周期用于执行存储器和I/O访问,中断响应周期用于响应由INTR引脚进入的外部中断请求信号。暂停和停机周期时,CPU处于暂停/停机状态。

2. 系统总线

系统总线是主机系统与外围设备之间的通信通道。在计算机主板上,系统总线表现为与扩展插槽线连接的一组逻辑电路和导线,与I/O扩充插槽相连。I/O插槽中可插入各式各样的扩充板卡,作为各种外设的适配器与外设连接,故这种总线也叫做I/O通道总线。

系统总线必须有统一的标准,以便按照这些标准设计各类适配卡。因此,我们实际上要讨论的总线就是系统总线,各种总线标准也主要是指系统总线的标准,包括与系统总线相连的插槽的标准。

系统总线包括地址信号线、数据信号线、控制信号线和电源、地线等。系统总线一般不

针对某一处理器而开发,标准化程度较高。

多年来,系统总线已成为计算机系统的通信瓶颈,这是因为:

- ① 微处理器运行速度不断提高,超过了系统总线提供的通信能力;
- ② 先进的存储器和外设接口极大地提高了数据传输率,如 SCSI 等;
- ③ 越来越多的外围设备需要高速的系统总线,如网络适配器、硬盘驱动器、图形适配器及各种专用设备等。

为了克服这些瓶颈,系统总线也随着微型计算机结构不断改进而不断发展,目前系统总线已制订了一些标准,例如 ISA 总线、EISA 总线、VESA 总线、AGP 总线、VME 总线、MCA 总线、PCI 总线等,而其中 ISA 总线、PCI 总线和 AGP 总线又是现在的微型计算机中最流行、最常见的系统总线。

ISA 总线 ISA(Industry Standard Architecture,工业标准体系结构)总线是由美国 IBM 公司推出的 8/16 位标准总线、数据传输率为 8 MB/s,主要用于 IBM - PC/XT、AT 及其兼容机上。

EISA 总线 EISA(Extended Industry Standard Architecture,扩展工业标准体系结构)总线是由 Compaq、HP、AST 等多家计算机公司联合推出的 32 位标准总线,数据传输率为 33 MB/s,适用 32 位微处理器。

MCA 总线 MCA(Micro Channel Architecture,微通道体系结构)是由美国 IBM 公司推出的 32 位标准总线。数据传输率为 40 MB/s,适用于 386、486 等微型计算机。

VESA 总线 由视频电子标准协会(Video Electronic Standard Association)联合另外多家公司共同推出的全开放通用的局部总线,它又称为 VL - BUS(Vesa Local Bus,简称 VL 总线)。VESA 的数据传输率为 133 MB/s。这种系统总线是针对 486 微型计算机开发的 32 位标准总线,可扩充至 64 位。一般用来连接图形显示接口,以缓解大量图像数据传输中的瓶颈问题。

PCI 总线 PCI(Peripheral Component Interconnect,外设互连)总线是由美国 Intel 公司推出的 32/64 位标准总线。PCI 总线是一种与 CPU 隔离的总线结构,并能与 CPU 同时工作。这种总线适应性强、速度快、数据传输率为 133 MB/s。适用于 Pentium 以上的微型计算机。

AGP 总线 AGP(Accelerated Graphics Port,加速图形端口)总线是一种专为提高视频带宽而设计的总线规范。其视频数据的传输速率可以从 PCI 的 133 MB/s 提高到 266 MB/s(×1 模式)、533 MB/s(×2 模式)、1.064 GB/s(×4 模式)和 2.128 GB/s(×8 模式)。严格地说,AGP 不能称为总线,因为它是点对点连接,即在控制芯片和 AGP 显示接口之间建立一个直接的通路,使 3D 图形数据不通过 PCI 总线,而直接送入显示子系统。这样就能突破由 PCI 总线形成的系统瓶颈。

3. 外部总线

外部总线用来提供输入/输出设备同系统中其他部件间的公共通信通路,标准化程度最

高,可用于各种型号的处理。也已制订了一些外部总线的标准,如小型计算机系统互连总线 SCSI、USB 总线、IEEE 1394 总线等,这些外部总线实际上应是主机与外设的接口。

4. 局部总线

局部总线用于主机内部特定子系统之间的紧密连接。在微型计算机内设置局部总线主要是为了提高 CPU 与高带宽占用部件(如显示卡)之间的传输速率。典型的局部总线包括 VESA 总线、AGP 总线等。

4.3 总线技术

4.3.1 总线的基本功能

接到总线上的部件有两种工作方式:主方式和从方式。部件工作于主方式时可以控制总线并启动信息传送,工作于从方式时只能按主部件的要求工作。有的部件既可工作于主方式,也可工作于从方式,但不能同时工作于两种方式。能以主方式工作的设备叫做总线主控设备。

从总线主部件申请使用总线到数据传送完毕的整个过程要经过几个步骤:总线请求—总线仲裁—寻址—传送数据—检错—发出数据出错信号。总线控制线路包括总线仲裁逻辑、驱动器和中断逻辑等。

总的来说,总线有以下几方面的基本功能:

① 总线数据传送 为使信息正确传送,防止丢失,需对总线通信进行定时,根据定时方式不同,大体可分为同步和异步两种数据传送方式。

② 总线仲裁控制 在总线上某一时刻只能有一个总线主部件控制总线,为避免多个部件同时发送信息到总线的矛盾,需要有总线仲裁机构。

③ 出错处理 数据经过总线进行传送过程中可能产生错误,在总线逻辑中应有错误检测电路或纠错电路。

④ 总线驱动 总线上连接的器件很多,因此要求总线要有一定的驱动能力,在计算机系统中通常采用三态总线驱动器来驱动总线。

4.3.2 总线的数据传送

数据在总线上传送时,送出数据的部件称为源部件,接收数据的部件称为目的部件。要确保在源部件和目的部件之间的数据传送协调一致,总线上的数据传送必须由定时信号控

制,定时信号使源部件和目的部件之间同步,实现两部件间的协调和配合。定时实现方式有三种:同步方式、异步方式和半同步方式。

1. 同步方式

总线上的数据传送用一个公共的时钟控制,发送和接收信号都在固定的时刻发出。图 4.6 是同步总线执行写操作时的定时图。数据在源端于某一固定的时刻在数据准备好信号 READY 控制下发出,在目的端由数据接收信号 ACK 控制接收。同步总线定时比异步总线定时的吞吐量大,因为在源端和目的端之间不需要有来往传送的“握手”控制信号,但延时时间 t_1 和 t_2 要根据接到总线上最慢的设备来设定。同步总线定时方法的缺点是源部件无法知道目的地部件是否已收到数据,目的地部件也无从知道源部件的数据是否已真正送到总线上。

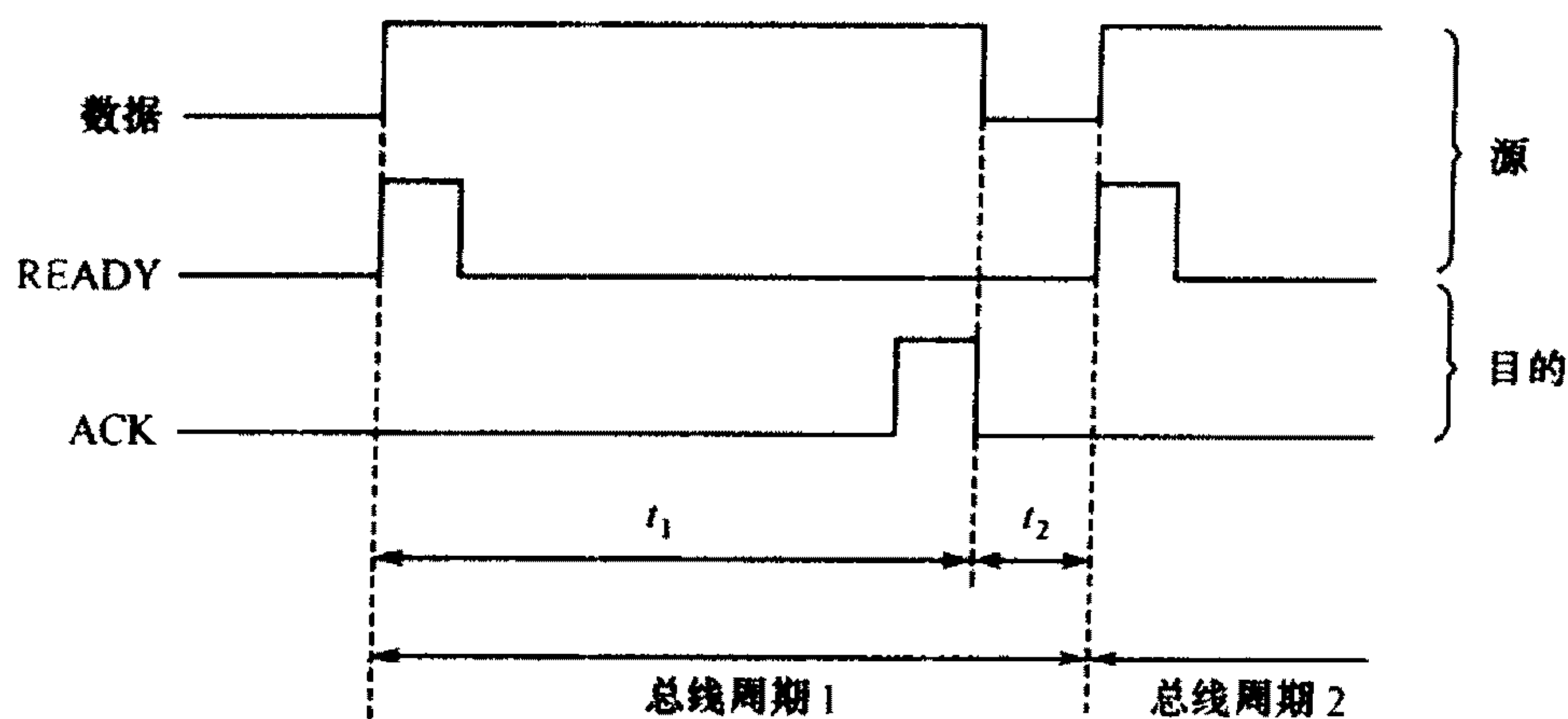


图 4.6 同步总线的定时图

2. 异步方式

异步定时方法中没有固定的时钟,定时序列中的每一步都要靠信号在源部件和目的地部件间的传送来实现。这些控制信号的传送要有相当可观的延迟时间。为减少信号传送的延迟,在异步方式中并不是每一步都靠信号传递来定时的,而是把某几步改用等待一段足够长的固定延迟时间来代替对方传送过来的信号。这种用固定延迟时间的信号叫作隐含信号,根据隐含信号的多少,可以把异步总线定时分为非互锁的、半互锁的和全互锁的三种方式。这里仅介绍非互锁异步定时方式,该方式下总线的定时时序如图 4.7 所示。在这个方式中,READY 信号和 ACK 信号的脉冲宽度设定为固定时间,即 t_2 和 t_4 为定值。数据送到总线上,经过延迟时间 t_1 后源端把 READY 信号升高,目的端收到 READY 信号后,接受总线上的数据,并经 t_3 时间后使 ACK 升高,以此通知源端,源端接受到 ACK 后,经 t_5 时间,从总线上撤去数据,再用 t_6 时间,使总线状态稳定,然后开始下一个总线周期。

这种方式的优点是任何速度的设备之间都能互相进行通信。缺点是延迟较大,而且当

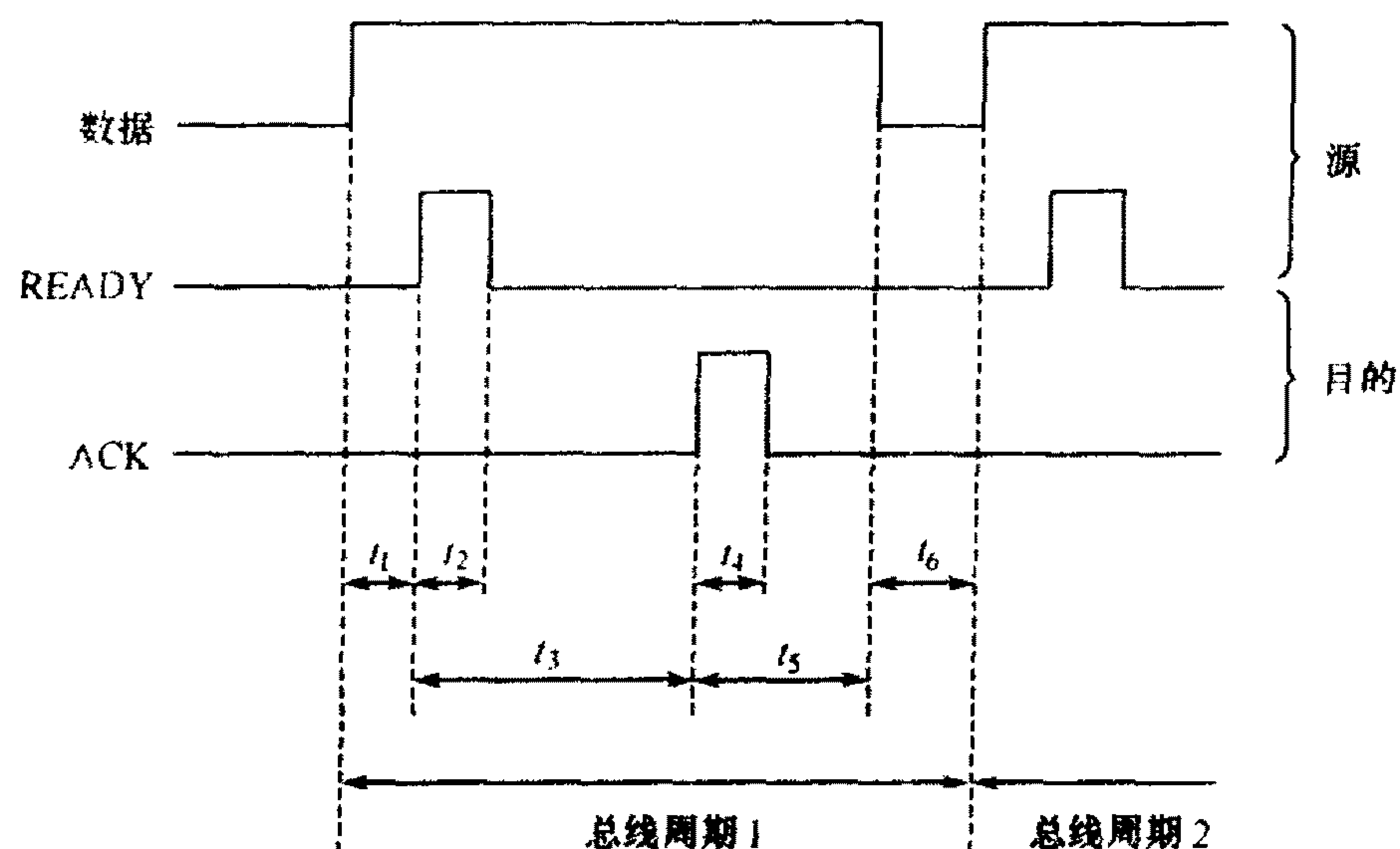


图 4.7 非互锁异步总线的定时时序图

源端速度和目的端速度差异很大时,有可能造成状态错误,使传送失败。

3. 半同步方式

采用半同步总线定时方法时,每个动作只能在固定时钟确定的一定时刻发生。控制信号间的间隔时间仍然可以是可变的,但间隔时间必须是时钟周期的整倍数。它的优点是减少了干扰信号的影响,因为只有在时钟信号上升沿和下降沿时发生的干扰信号才会被误认为是定时信号。它的缺点是定时信号的响应时间都必须是时钟周期的整倍数,加长了时间间隔。

4.3.3 总线的仲裁控制

总线仲裁也称为总线判优。总线是多个部件所共享的,在总线上某一时刻只能有一个总线主控部件控制总线,为了正确地实现多个部件之间的通信,避免各部件同时往总线发送信息造成的冲突,必须要有一个总线仲裁(控制)机构,对总线的使用进行合理的分配和管理。

当总线上的一个部件要与另一个部件进行通信时,首先应该发出总线请求信号。在某一时刻,可能有多个部件同时要求使用总线,总线控制机构根据一定的判决原则,决定首先由那个部件使用总线。只有获得了总线使用权的部件,才能开始传送数据。

根据总线控制部件的位置,控制方式可以分成集中方式与分散方式两类。总线控制逻辑集中在一处的,称为集中式总线控制。总线控制逻辑分散在总线各控制部件中的,称为分散式总线控制。以下介绍的是集中式总线控制方式。

1. 链式查询方式

链式查询方式如图 4.8 所示。图中所示的总线控制部件在单总线系统和三总线系统中常常是 CPU 的一部分。在双总线系统的 I/O 总线中,它是通道的一部分。链式方式,除一般数据总线(DB)和地址总线(AB)外,主要还有 3 根控制线:

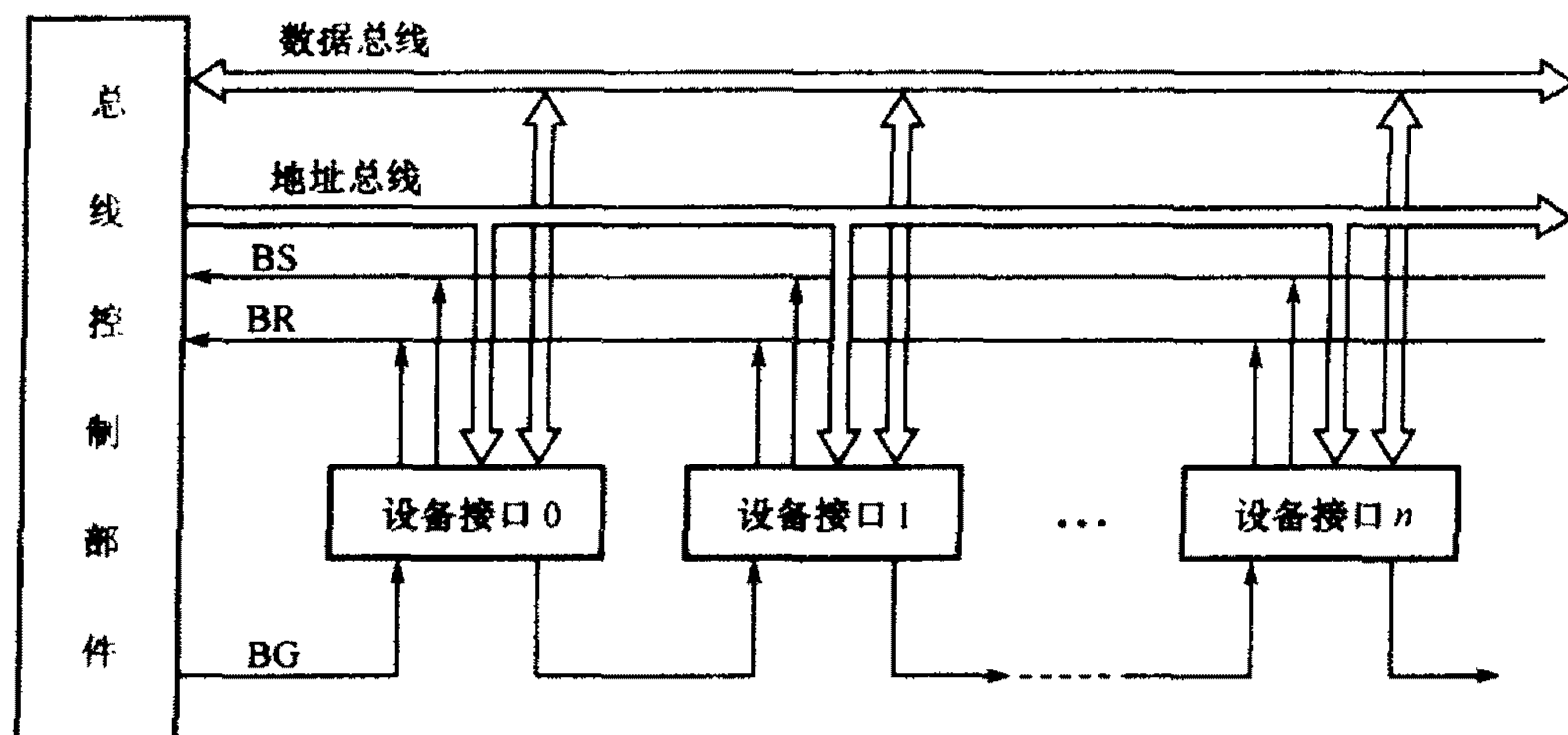


图 4.8 链式查询方式

- ① 总线忙信号 BS 信号有效时,表示总线正被某外设使用;
- ② 总线请求信号 BR 信号有效时,表示至少有一个外设请求使用总线;
- ③ 总线回答信号 BG 该信号有效,表示总线控制部件响应了外设的总线请求。

链式查询方式的主要特征是,总线回答信号 BG 的传送是串行地从一个 I/O 接口送到下一个 I/O 接口。假如 BG 到达的接口无总线请求,则继续往下传。假如 BG 到达的接口有总线请求,BG 信号便不再往下传,这意味着该 I/O 接口就获得了总线使用权。

显然,在查询链中离总线控制器最近的设备具有最高优先权,离总线控制器越远,优先权越低。因此,链式查询是通过接口的优先权排队电路来实现的。

链式查询方式的优点是,只用很少几根线就能按一定优先次序实现总线控制,并且这种链式结构很容易扩充设备。链式查询方式的缺点是对询问链的电路故障很敏感,如果第 i 个设备的接口中有关键的电路有故障,那么第 i 个以后的设备都不能进行工作。另外查询链的优先级是固定的。如果优先级高的设备出现频繁的请求时,那么优先级较低的设备可能长期不能使用总线。

2. 计数器查询方式

计数器查询方式原理如图 4.9 所示。总线上任何设备要求使用总线时,都通过 BR 线发出总线请求。总线控制部件接到总线请求信号后,在 BS 线为 0 的情况下让计数器开始计

数,计数值通过一组设备地址线发向各设备。每个外设接口都有一个设备地址判别电路,当设备地址线上的计数值与请求使用总线的设备地址一致时,该设备就获得了总线使用权,并置 BS 线为 1,此时中止计数查询。

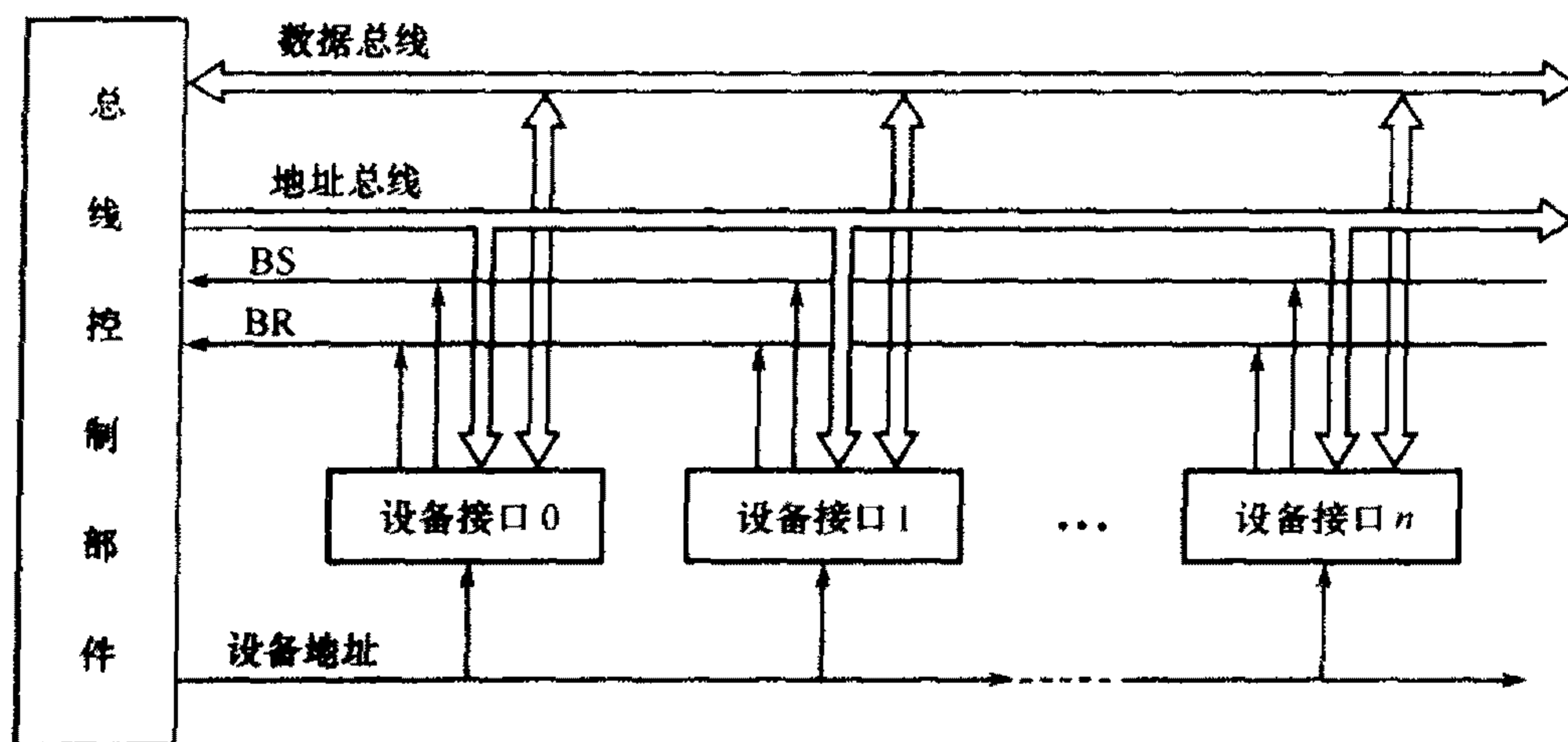


图 4.9 计数式查询方式

每次计数可以从 0 开始,也可从上次计数的中止点开始。如果从 0 开始,各设备的优先次序与链式查询相同,优先级的顺序是固定的。如果从中止点开始,则每个设备使用总线的优先级别是相等的。计数器的初值也可以用程序来设置,这就可以方便地改变优先次序,显然这种灵活性是以增加相应的控制线数为代价的。

3. 独立请求方式

独立请求方式原理如图 4.10 所示。在独立请求方式中,每一个设备均有独立的总线请求线 BR_i 和总线回答线 BG_i 。当设备要求使用总线时,便发出总线请求信号 BR_i 。总线控制部件中的优先排队电路根据一定的优先次序决定首先响应哪个设备的请求,然后向该设备发出总线应答信号 BG_i ,表示允许该设备占用总线。

独立请求方式的优点是响应时间快,即为确定优先响应设备所花费的时间少,不用逐个查询设备,然而这是以增加控制线数为代价的。在链式查询中仅用两根线确定总线使用权属于哪个设备;在计数查询中大致用 $\log_2 n$ 根线,其中 n 是允许接纳的最大设备数;而独立请求方式需采用 $2n$ 根线。

独立请求方式对优先次序的控制也是相当灵活的。它可以预先固定,如让 BR_0 优先级最高、 BR_1 次之…… BR_n 最低;也可以通过程序来改变优先次序,或采用屏蔽某个请求的办法,不响应来自与当前处理无关的设备的请求。

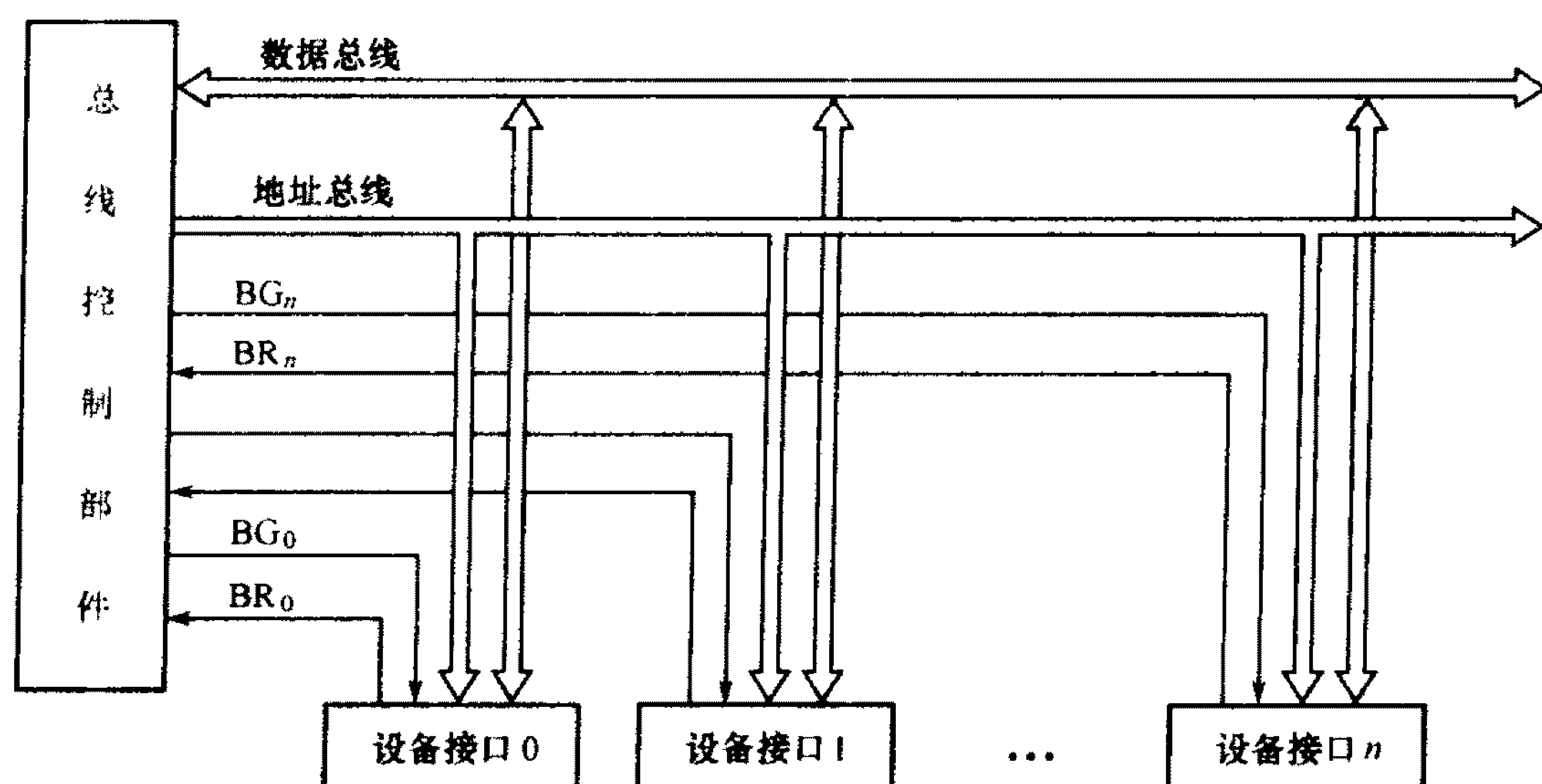


图 4.10 独立请求方式 1

4.3.4 总线驱动及出错处理

1. 总线驱动

在计算机系统中,总线上连接的设备接口很多,每个接口电路都要从总线上吸收电流,故常采用三态总线驱动器来驱动总线。但总线驱动器的驱动能力是有限的,因此在扩充外设接口时要注意,通常一个模块或部件限制为 1~2 个负载,而且必须是低功耗的负载。为减轻总线上的负载,在设备接口电路中通常需要采用缓冲器来进行隔离。图 4.11 所示为连接在 ISA 总线上的一个 32 位并行输出接口电路。在此电路中,74LS244 缓冲器用于减轻数据总线上的负载。如果没有 74LS244 缓冲器,则此接口电路将在数据总线上加上 4 个负载。若所有接口都往总线上加上如此沉重的负载,则系统将无法进行工作。

2. 出错处理

数据传送过程中可能产生错误,解决的方法是在传输的数据中增加一些冗余位,使冗余位与传送的数据具有某种特殊的关系,例如使数据中 1 的个数为偶数,这样接收部件中的错误校验电路就可以检查出接收的数据是否出错。若这种特殊关系存在,表示接收的数据正确,若这种特殊关系不存在,表示接收的数据出错。发现错误后,如何去处理错误通常有两种方法:当总线控制器和设备接口中的总线接口部件有自动纠错电路时,纠错电路可以根据错误的状态用某种算法自动纠正错误;若部件中无自动纠错电路,则可在发现错误后发出“数据出错”信号让 CPU 来进行错误处理,CPU 发出中断请求信号,响应中断后转入错误处理程序来处理异常情况。

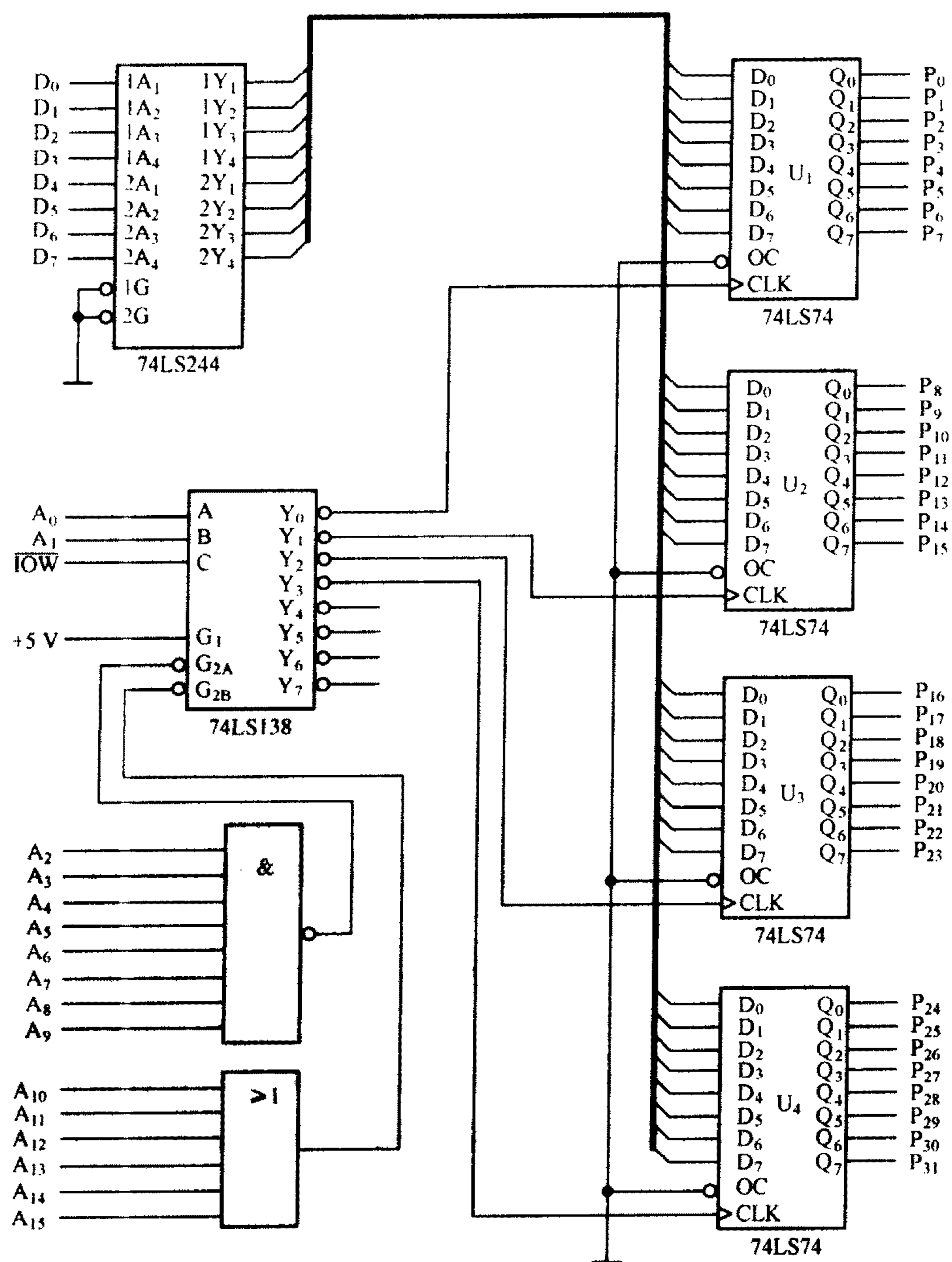


图 4.11 连接到 8 位 ISA 总线上的 32 位并行输出接口

4.3.5 总线的性能指标

1. 总线的带宽

总线的带宽指的是单位时间内总线上可传送的数据量,即每秒钟传送多少字节。与总

线带宽密切相关的两个概念是总线的位宽和总线的工作频率。

2. 总线的位宽

总线的位宽指的是总线能同时传送的数据位数,即通常所说的 16 位、32 位、64 位等总线宽度的概念。在工作频率一定的条件下,总线的带宽与总线的位宽成正比。

3. 总线的工作频率

总线的工作频率也称为总线的时钟频率,以 MHz 为单位。它是指用于协调总线上的各种操作的时钟信号的频率。工作频率越高,总线工作速度越快,即总线带宽越宽。

总线带宽、总线位宽、总线工作频率三者之间的关系就像高速公路上的车流量、车道数和车速的关系。车流量取决于车道数和车速,车道数越多、车速越快则车流量越大;同样,总线带宽取决于总线位宽和工作频率,总线位宽越宽、工作频率越高则总线带宽越大。当然,单方面提高总线的位宽或工作频率也能部分提高总线的带宽,但容易达到各自的极限。

总线带宽的计算公式如下:

$$\text{总线带宽} = (\text{总线位宽}/8) \times \text{总线工作频率}(\text{MB/s})$$

例如,以 66 MHz 频率工作的 32 位总线,若每个时钟传送一次数据,则

$$\text{总线带宽} = 32/8 \times 66 = 264 \text{ MB/s}$$

4.4 常用系统总线

在国际化生产流行的今天,一台计算机往往不再是由单一的企业生产出来。而是将计算机中的各部件交给不同的专业化生产厂家分别生产,然后由组装厂组装。这样做的目的主要是为了降低成本、提高生产率和产品的质量。为了将不同厂家生产的各部件组装成一台完整的计算机,就需要各厂家按一定的标准进行生产。特别是系统总线,由于外设接口卡都要通过它接入系统,所以总线标准的制订更显重要。目前系统总线已制订了一些标准,例如 S-100 总线、STD 总线、Multibus 总线、ISA 总线、EISA 总线、VESA 总线、AGP 总线、VME 总线、MCA 总线、PCI 总线等。需要注意的是,在计算机行业中,往往是先出产品,后定标准,谁的产品市场占有率高,谁就拥有这个产品的标准制定权,或称其为事实上的标准。上述这些总线标准中,有相当一部分都是因为市场占有率高而形成了事实上的标准。本节主要介绍 ISA 总线、EISA 总线、PCI 总线以及 AGP 总线。

4.4.1 系统总线标准的内容

系统总线用来连接各子系统的插件板,即各插件板的插座之间是用系统总线连接的。

为使各插件板的插座之间具有通用性,使一个系统中的各插件板可以插在任何一个插座上,方便用户的安装和使用,另外还希望不同厂家的插件板可以互连、互换,这样就必须有一个规范化的可通用的系统总线。为了兼容,还要求插件的几何尺寸相同,插座的针数相同,插座中各针的定义相同,信号的电平和工作的时序相同。系统总线通常为 50 ~ 100 根信号线,这些信号线可分为 5 个主要类型:

- ① 数据线 决定数据宽度;
- ② 地址线 决定直接选址范围;
- ③ 控制线 包括控制、时序和中断线,决定总线功能和适应性的好坏;
- ④ 电源线和地线 决定电源的种类及地线的分布和用法;
- ⑤ 备用线 留给厂家或用户自己定义。

有关这些信号线的标准主要涉及到如下几个方面:

- ① 信号的名称;
- ② 信号的定时关系;
- ③ 信号的电平;
- ④ 连接插件的几何尺寸;
- ⑤ 连接插件的电气参数;
- ⑥ 引脚的定义、名称和序号;
- ⑦ 引脚的个数;
- ⑧ 引脚的位置;
- ⑨ 电源及地线。

微型计算机自问世以来,从 8 位机到 16 位机、32 位机一直发展到了 64 位机。为了适应数据宽度的增加和系统性能的提高,依次推出并采用的系统总线标准有 XT 总线、ISA 总线、EISA 总线和 PCI 总线。

4.4.2 ISA 和 EISA 总线

1. ISA 总线

ISA 数据传输率(即总线带宽)为 8 MB/s 或 16 MB/s,主要用于 IBM - PC/XT、AT 及其兼容机上。在早期的 PC 机中,ISA 总线应用非常广泛,大多数计算机主板上只提供 ISA 插槽。随着技术的进步,ISA 总线已逐渐被淘汰。现在,大多数 PC 机主板上只保留了一个 ISA 插槽,有些较新型的主板上甚至已不再提供 ISA 插槽。

ISA 插槽有 62 根引脚,用于插入 8 位的插卡;8/16 位的扩展插槽除了具有一个 8 位 62 线的连接器外,还有一个附加的 36 线连接器,这种扩展 I/O 插槽既可支持 8 位的插卡,也可

支持 16 位插卡。ISA 总线的主要性能指标如下：

- ① I/O 地址空间为 0100H ~ 03FFH;
- ② 24 位地址线可直接寻址的内存容量为 16 MB;
- ③ 8/16 位数据线;
- ④ 62 + 36 引脚;
- ⑤ 最大位宽 16 位(bit);
- ⑥ 最高时钟频率 8 MHz;
- ⑦ 最大稳态传输率 16 MB/s;
- ⑧ 中断功能;
- ⑨ DMA 通道功能;
- ⑩ 开放式总线结构,允许多个 CPU 共享系统资源。

ISA 插槽外形如图 4.12 所示。图中深色插槽的左半部分是 8 位 ISA 总线插座,较窄的右半部分是 16 位 ISA 总线扩充插座。8 位 ISA 总线接口卡只使用左半部分的总线信号,16 位 ISA 总线接口卡要使用右边的部分信号。

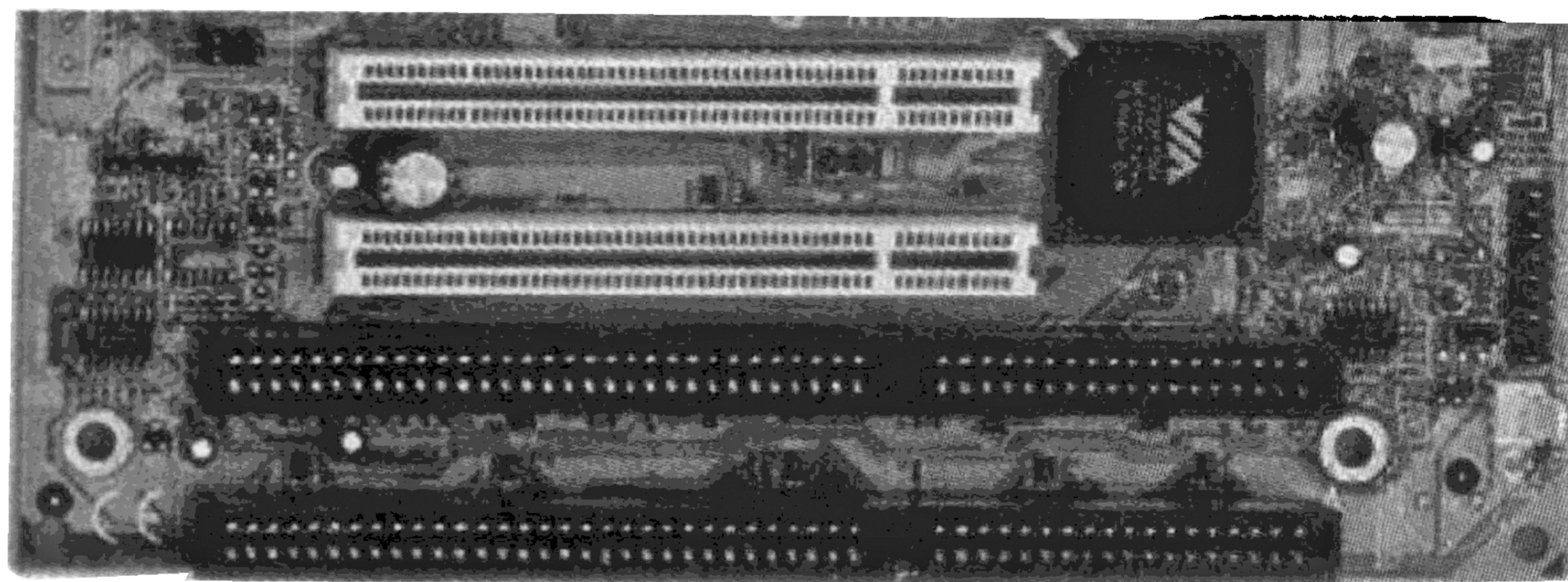


图 4.12 ISA 插槽外形

8 位 ISA 总线的信号全部连接到一个 62 针的插座上,分成 A、B 两排(接触面),每排 31 针,其中数据线 8 根,地址线 20 根,可接收 6 路中断请求,3 路 DMA 请求,此外还包括时钟、电源线和地线。标有 A1 ~ A31 及 B1 ~ B31 的 62 线插槽用于插入 8 位数据宽度插卡的插槽。C1 ~ C18 和 D1 ~ D18 为 16 位 ISA 总线增加的 36 线插槽,它和 62 线插槽一起供 16 位插卡使用。16 位 ISA 总线在 8 位 ISA 总线的基础上将数据线扩充到 16 根,地址线扩充到 24 根。可支持 15 级中断和 7 个 DMA 通道。

2. EISA 总线

ISA 总线对于 286 和 386SX 等微型计算机系统来说是方便的,但对于 386DX 以上档次

具有 32 位地址和数据宽度的微型计算机系统来说,因其数据总线和地址总线宽度不够而影响了 32 位微处理器性能的发挥。为此,Compaq、HP、AST 等九家公司联合起来在 ISA 的基础上于 1988 年推出了为 32 位微型计算机设计的“扩展工业标准结构”(Extended Industry Standard Architecture),即 EISA 总线。

EISA 在结构上与 ISA 有良好的兼容性,保护了厂商和用户的权益;同时又充分发挥和利用 32 位微处理机的功能,使之在图形技术、光存储器、分布处理、网络、数据处理等需要高速处理能力的地方发挥作用。

EISA 总线可支持 80486 及以前的 X86 CPU,但它不支持 Pentium 及以后的各类新型微处理器,因为从 Pentium 开始的处理器已使用 64 位数据总线,而 EISA 总线并不支持 64 位数据总线。

EISA 总线主要具有以下几个特点:

① 可支持 CPU 等总线主控制器的 32 位寻址能力和 16 位、32 位的数据传送能力,对数据宽度具有变换功能。

② 扩展和增强了 DMA 仲裁和传输能力,使 DMA 的数据传输率最高可达 33 MB/s。EISA 总线与系统主板交换数据的速率比 ISA 快 4 倍。

③ 可通过软件实现系统主板和扩充板的自动配置功能,无需借助 DIP 开关。EISA 系统和 ISA 系统不同,任何挂接到 EISA 总线上的接口卡必须经配置后才能被系统所识别。EISA 系统的配置是通过运行专门的软件(EISA Configuration Software, ECS),将 EISA 总线上各设备的情况记录在一片 8 Kb×8 的 SRAM 中,使系统对这些 EISA 资源进行有效的管理和使用。这个 SRAM 如同 ISA 系统中的 CMOS 一样,需要使用电池供电,以保持其中内容不丢失,一旦遇到信息丢失,或有新的 EISA 设备加入时,需要重新进行配置。

④ 可管理多个总线主控制器,并使用突发方式对系统存储器进行读/写访问。两个总线主控制器之间通过 EISA 总线也可以进行数据交换。另外,EISA 的总线主控制器可不占用 DMA 通道。而 ISA 总线对于每一个总线主控制器都要使用一个 DMA 通道。

⑤ 可用程序来控制中断请求采用边沿触发或电平触发方式。

⑥ EISA 总线插槽既可插 ISA 插卡,又可插 EISA 插卡。在插 EISA 卡时使用 32 位数据线,可达到 33 MB/s 的传输率。

EISA 的主要性能指标有:

① 开放式结构 EISA 和 ISA 兼容,现有的 ISA 扩充板可以用于 EISA 总线上;

② 32 位地址宽度,直接寻址范围为 4 GB;

③ 32 位数据线;

④ 最大时钟频率 8.3 MHz;

⑤ 最大传输率 33 MB/s。

EISA 总线连接器(插头、插槽)的尺寸大小与 ISA 总线连接器大致相同,所不同的是插头和插槽都分成两层:上层为 ISA 连接点,其结构和引脚信号定义均与 ISA 总线完全兼容,这就使 ISA 标准扩充卡可方便地用于 EISA 系统中,就像在 ISA 系统中使用一样。下层为 EISA 连接点,用于扩展方式,它同上层联合起来构成 32 位 EISA 总线。

对于不同的 EISA 总线扩充槽,它的 I/O 地址是不同的,如对于第 6 个总线扩充槽,它的 I/O 地址为 1XXX6;但为保持与 ISA 的兼容,每个 EISA 总线扩充槽都支持 0H~3FFH 的 I/O 地址访问。

4.4.3 PCI 总线

PCI 总线是一种与 CPU 隔离的总线结构,并能与 CPU 同时工作。这种总线适应性强、速度快、数据传输率为 133 MB/s。适用于 Pentium 以上的微型计算机。

1. 概述

随着图形处理技术和多媒体技术的广泛应用,计算机处理的信息除了传统的文字、图形信息外,还包括了音频和视频信息。与传统微型计算机处理的文本信息和简单的图形信息相比,音频和视频信息具有实时性强、复合性高、信息量大的特点。实时性是指播放声音、图像和视频时要求声音或画面能连续、平滑地变化。复合性是指在播放声音或图像时,声像必须保持同步与协调。与文字信息相比,音频与视频的信息量要大的多。仅仅存储 1 分钟的声音其信息量可达到 2~6 MB,存储 1 秒钟的视频图像信息量更高达 9~10 MB。这些特点,要求现代的微型计算机应具有更快的处理速度、更大的存储空间和更高的总线带宽。事实上,微型计算机中的一些关键部件,如 CPU、内存、显示卡、硬盘等经过多年来的不断改进,在性能上有了很大的提高,CPU 的速度为几百 MHz 以上,硬盘与硬盘控制器之间的数据传输率可达 33 MB/s 以上,网络卡的数据传输率达到约 100 MB/s,图形控制器和显示器之间的数据传输率达到 200 MB/s 以上。通常认为 I/O 总线的速度应为外设速度的 3~5 倍。显然原有的 ISA、EISA 已远远不能适应要求,而成为整个系统的主要瓶颈。需求带动了技术的进步,在这种情况下,1991 年下半年,由 Intel 公司首先提出的新的 PCI 总线标准便应运而生了。

2. PCI 总线的主要性能和特点

PCI 是一种先进的局部总线,不依附于某个具体处理器,其总线插槽的外形如图 4.13 所示。从结构上看,PCI 是在 CPU 和原来的 ISA 系统总线之间插入的另一级总线,具体由一个桥接电路实现对这一层的管理,并实现上下之间的接口以协调数据的传送。管理器提供了信号缓冲,使之能支持 10 种类型的外设接口,并能在高时钟频率下保持高性能。PCI 总线也支持总线主控技术,允许智能设备在需要时取得总线控制权,以加速数据传送。

一个 PCI 接口包括一系列的寄存器,它们位于 PCI 接口上的一个小容量的存储器中,其

中包含了 PCI 接口的信息。根据这些寄存器中的信息,计算机就可以把 PCI 接口自动配置到系统中,这个特性被称为即插即用(PnP)特性,这也是 PCI 总线在最新的计算机系统中变得如此流行的原因之一。PCI 总线的主要性能与特点如下:

- ① 总线时钟频率 33 MHz/66 MHz;
- ② 最大数据传输速率在时钟频率为 33 MHz 时为 133 MB/s(32 位)或 266 MB/s(64 位);
- ③ 时钟同步方式;
- ④ 总线宽度 32 位/64 位;
- ⑤ 能自动识别外设(即插即用功能);
- ⑥ 具有与处理器和存储器子系统完全并行操作的能力;
- ⑦ 支持 64 位寻址能力;
- ⑧ 完全的多总线主控能力。

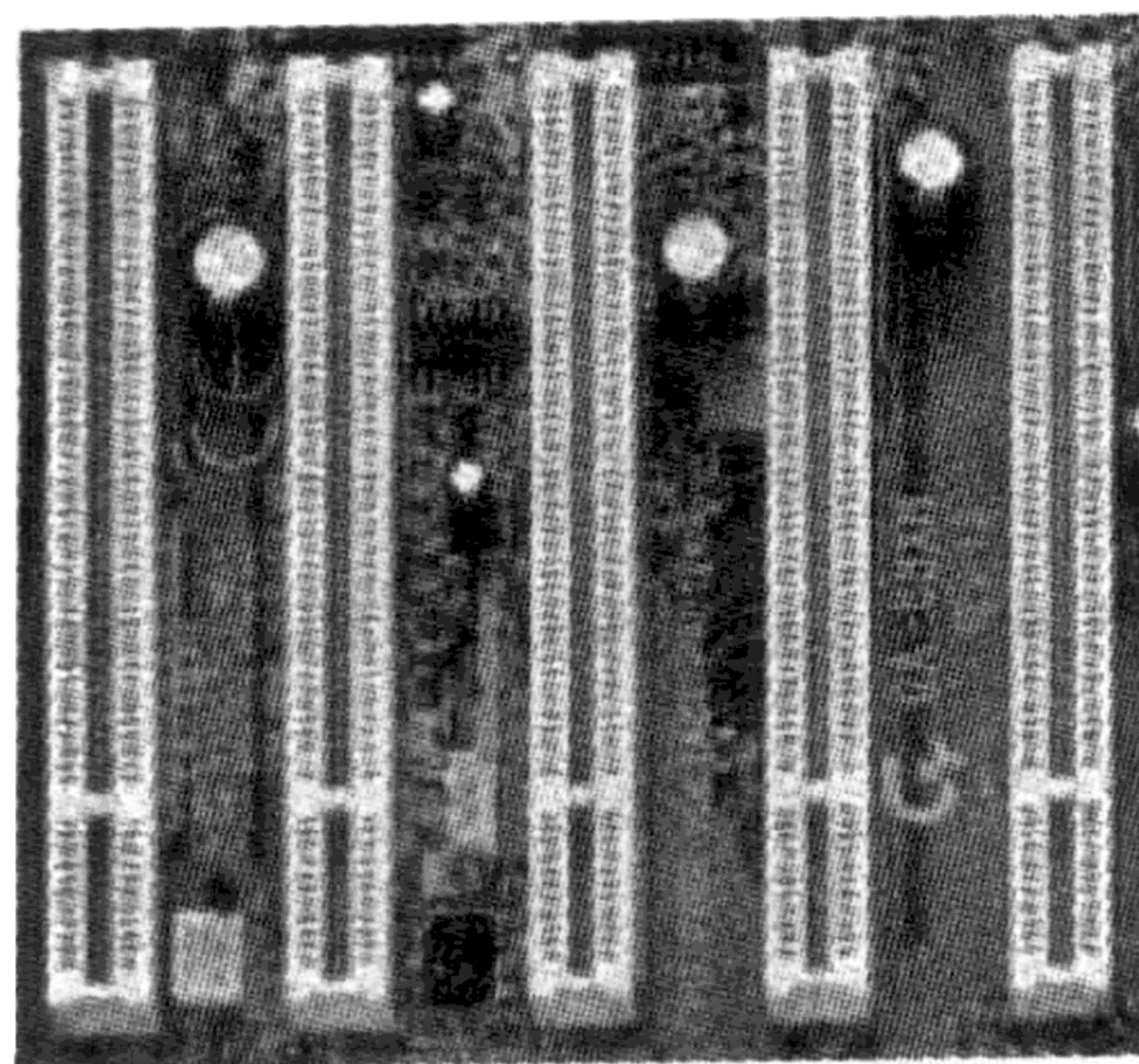


图 4.13 PCI 总线插槽外形图

3. PCI 总线引脚

PCI 总线包含了 32 位(或 64 位)数据总线和 32 位(或 64 位)的地址总线。地址总线和数据总线是多路复用的,它们在连接器上标识为 AD0 ~ AD63。32 位 PCI 总线具有连接引脚号 1 ~ 62,而 64 位 PCI 总线具有连接引脚号 1 ~ 94。表 4-1 给出了 PCI 总线的引脚图。

表中 63 ~ 94 号引脚只用于 64 位 PCI 总线, + V(I/O)在 3.3 V 主板上为 + 3.3 V,在 5 V 主板上为 + 5 V。

4. PCI 总线结构连接方式

PCI 总线的基本连接方式如图 4.14 所示。注意,微处理器的总线是单独的,并独立于 PCI 总线。从图中可以看到 CPU 总线(即前端总线)和 PCI 总线由桥接电路(习惯上称为北桥芯片或 GMCH)相连。北桥芯片中除了含有桥接电路外,还有 Cache 控制器和 DRAM 控制器等其他控制电路。PCI 总线上挂接高速设备,如图形控制器、IDE 设备或 SCSI 设备、网络控制器等。PCI 总线和 ISA/EISA 总线之间也通过桥接电路(习惯上称为南桥芯片或 ICH)相连,ISA/EISA 上挂接传统的慢速设备,继承原有的资源。

此外,PCI 总线还有其他一些连接方式,如双 PCI 总线方式、PCI 到 PCI 方式、多处理器服务器方式等。

表 4-1 PCI 总线引脚信号

引脚	焊接面	元件面	引脚	焊接面	元件面	引脚	焊接面	元件面
1	- 12 V	TRST	33	C/BE ₂	+ 3.3 V	65	C/BE ₆	C/BE ₅
2	+ TCK	+ 12 V	34	GND	FRAME	66	C/BE ₄	+ V(I/O)
3	GND	TMS	35	IRDY	GND	67	GND	PAR ₆₄
4	TDO	TDI	36	+ 3.3 V	TRDY	68	AD ₆₃	AD ₆₂
5	+ 5 V	+ 5 V	37	DEVSEL	GND	69	AD ₆₁	GND
6	+ 5 V	INTA	38	GND	STOP	70	+ 5 V(I/O)	AD ₆₀
7	INTB	INTC	39	LOCK	+ 3.3 V	71	AD ₅₉	AD ₅₈
8	INTD	+ 5 V	40	PERR	SDONE	72	AD ₅₇	GND
9	PRSNT ₁	-	41	+ 3.3 V	SBO	73	GND	AD ₅₆
10	-	+ V(I/O)	42	SERR	GND	74	AD ₅₁	AD ₅₄
11	PRSNT ₂	-	43	+ 3.3 V	PAR	75	AD ₅₃	+ V(I/O)
12	-	-	44	C/BE ₁	AD ₁₅	76	+ V(I/O)	AD ₅₂
13	-	-	45	AD ₁₄	+ 3.3 V	77	AD ₅₁	AD ₅₀
14	-	-	46	GND	AD ₁₃	78	AD ₄₉	GND
15	GND	RST	47	AD ₁₂	AD ₁₁	79	+ V(I/O)	AD ₄₈
16	CLK	+ V(I/O)	48	AD ₁₀	GND	80	AD ₄₇	AD ₄₆
17	GND	GNT	49	GND	AD ₉	81	AD ₄₅	GND
18	REQ	GND	50	-	-	82	GND	AD ₄₄
19	+ V(I/O)	-	51	-	-	83	AD ₄₃	AD ₄₂
20	AD ₃₁	AD ₃₀	52	AD ₈	C/BE ₀	84	AD ₄₁	+ V(I/O)
21	AD ₂₉	+ 3.3 V	53	AD ₇	+ 3.3 V	85	GND	AD ₄₀
22	GND	AD ₂₈	54	+ 3.3 V	AD ₆	86	AD ₃₉	AD ₃₈
23	AD ₂₇	AD ₂₆	55	AD ₅	AD ₄	87	AD ₃₇	GND
24	AD ₂₅	GND	56	AD ₃	GND	88	+ V(I/O)	AD ₃₆
25	+ 3.3 V	AD ₂₄	57	GND	AD ₂	89	AD ₃₅	AD ₃₄
26	C/BE ₃	IDSEL	58	AD ₁	AD ₀	90	AD ₃₃	GND
27	AD ₂₃	+ 3.3 V	59	+ V(I/O)	+ V(I/O)	91	GND	AD ₃₂
28	GND	AD ₂₂	60	ACK ₆₄	REQ ₆₄	92	-	-
29	AD ₂₁	AD ₂₀	61	+ 5 V	+ 5 V	93	-	GND
30	AD ₁₉	GND	62	+ 5 V	+ 5 V	94	GND	-
31	+ 3.3 V	AD ₁₈	63	-	GND			
32	AD ₁₇	AD ₁₆	64	GND	C/BE ₇			

5. PCI 总线周期

PCI 的操作有两种模式：

① 正常模式 地址和数据交替使用 AD 总线。首先发送的是地址信号,接着就是数据的读/写。正常模式一次传输过程需要 2~3 个时钟周期(地址周期 + 写周期;地址周期 + 读周期 + 写周期)。因此对一个 32 位宽的数据总线来说,最大写数据传输速度只有 66 MB/s,

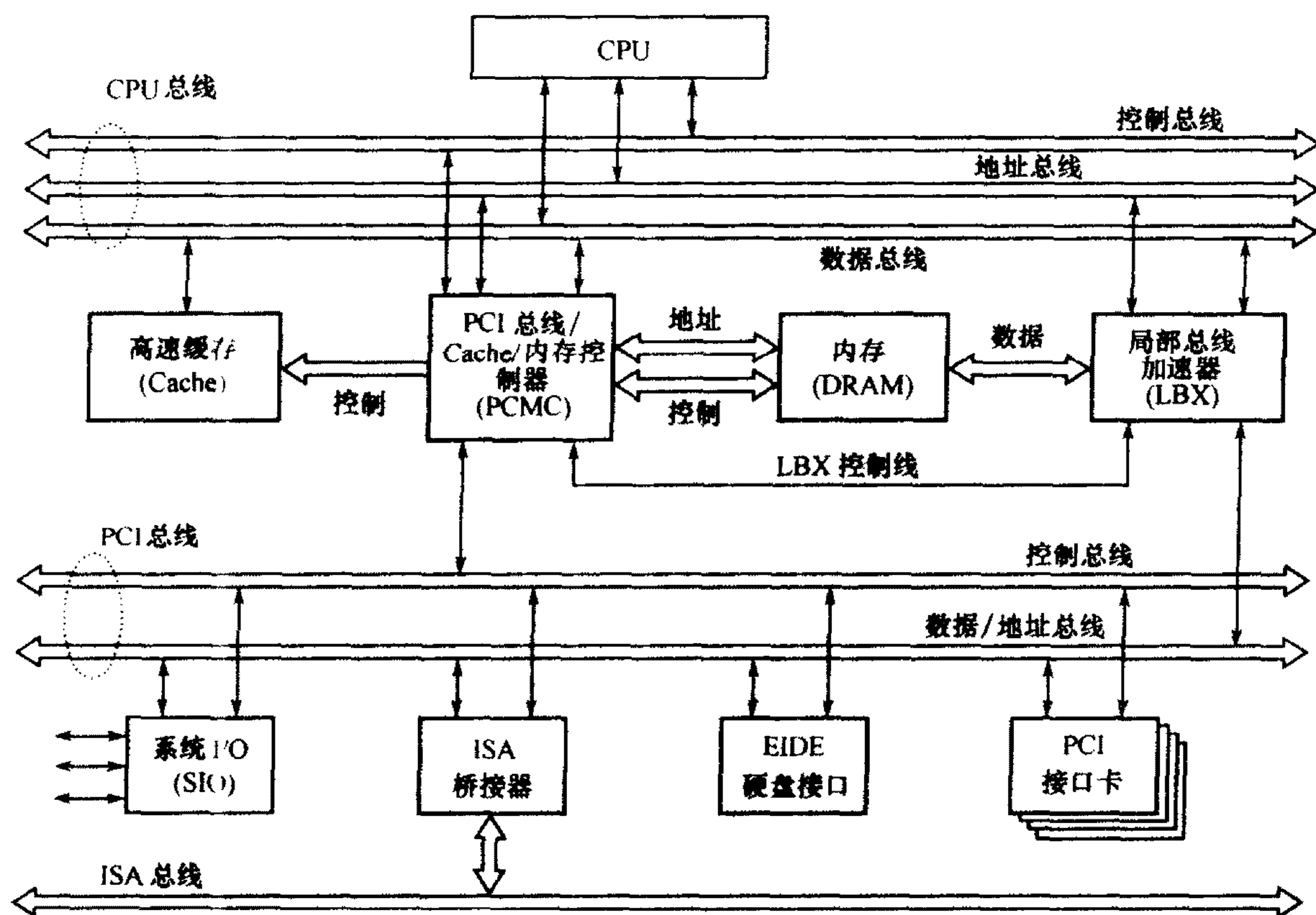


图 4.14 PCI 总线的连接方式

而最大的读数据传输速度只有 44 MB/s。

② 突发模式 在这种模式下,主设备先发出一个起始地址,接着是一系列隐含地址(地址顺序增量)的数据信号。这样,如果传输的是具有连续地址的内存块,那么数据传输速度最高可达到 133 MB/s(32 位微处理器)或是 266 MB/s(64 位微处理器)。

图 4.15 给出了 32 位 PCI 总线在突发模式下的时序图。

在时序图的第一个时钟周期中,主控设备把地址放到 AD 总线上,把对目标设备的命令放到 $\overline{C/BE}$ (命令/字节选通)引脚上。 $\overline{C/BE}$ 引脚上的状态标识了不同种类的命令,见表 4-2。

PCI 总线允许任何两个设备之间进行对话,一个设备不需要经过处理器就可以与另一个设备进行通信。启动通信的设备被称为启动(主)设备,而从设备被称为目标设备。在突发模式下,写周期的操作顺序是:

① 在寻址阶段,启动设备激活 \overline{FRAME} 信号,把命令放到命令/字节选通信号线($\overline{C/BE}_3 \sim \overline{C/BE}_0$)上,把地址放到地址/数据引脚($AD_0 \sim AD_{31}$)上。接着总线利用命令/字节选通信号($\overline{C/BE}_3 \sim \overline{C/BE}_0$)来传输指定长度的字节。

② 目标设备将 \overline{TRDY} (目标设备准备就绪)信号变低,表示目标设备可以接收总线上的数据。另外,启动设备将 \overline{IRDY} 变低,表示它已经准备好向 PCI 桥接器发送数据。

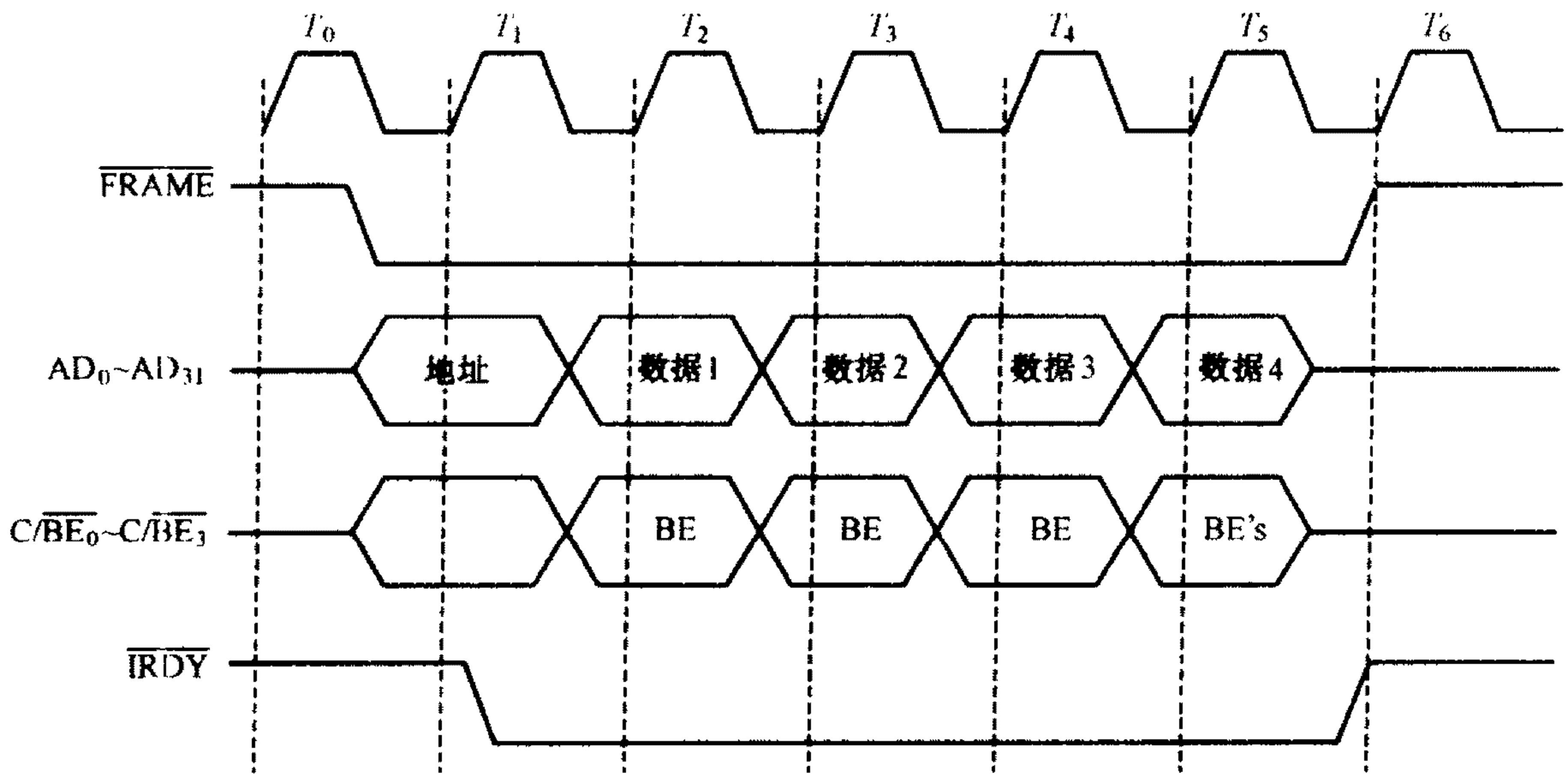


图 4.15 PCI 总线的基本突发模式时序图

表 4-2 PCI 总线命令

C/\overline{BE}_3	C/\overline{BE}_2	C/\overline{BE}_1	C/\overline{BE}_0	命 令
0	0	0	0	INTA 序列
0	0	0	1	特殊周期
0	0	1	0	I/O 读
0	0	1	1	I/O 写
0	1	1	0	存储器读
0	1	1	1	存储器写
1	0	1	0	配置存储器读
1	0	1	1	配置存储器写
1	1	0	0	内存多路读
1	1	0	1	双地址存取
1	1	1	0	线性存储器读
1	1	1	1	带刷新的存储器写

③ 在多个命令/字节选通信号的驱动下,传输可以连续进行。启动设备可以通过将 \overline{IRDY} 变高来中断传输,而目标设备也可以通过将 \overline{TRDY} 变高来中断传输。

④ 将 \overline{FRAME} 变高,传输结束。

读周期与写周期很类似,只不过目标设备利用 \overline{TRDY} 信号来表示总线上的数据是有效的。PCI 总线的数据交换过程见图 4.16 所示。

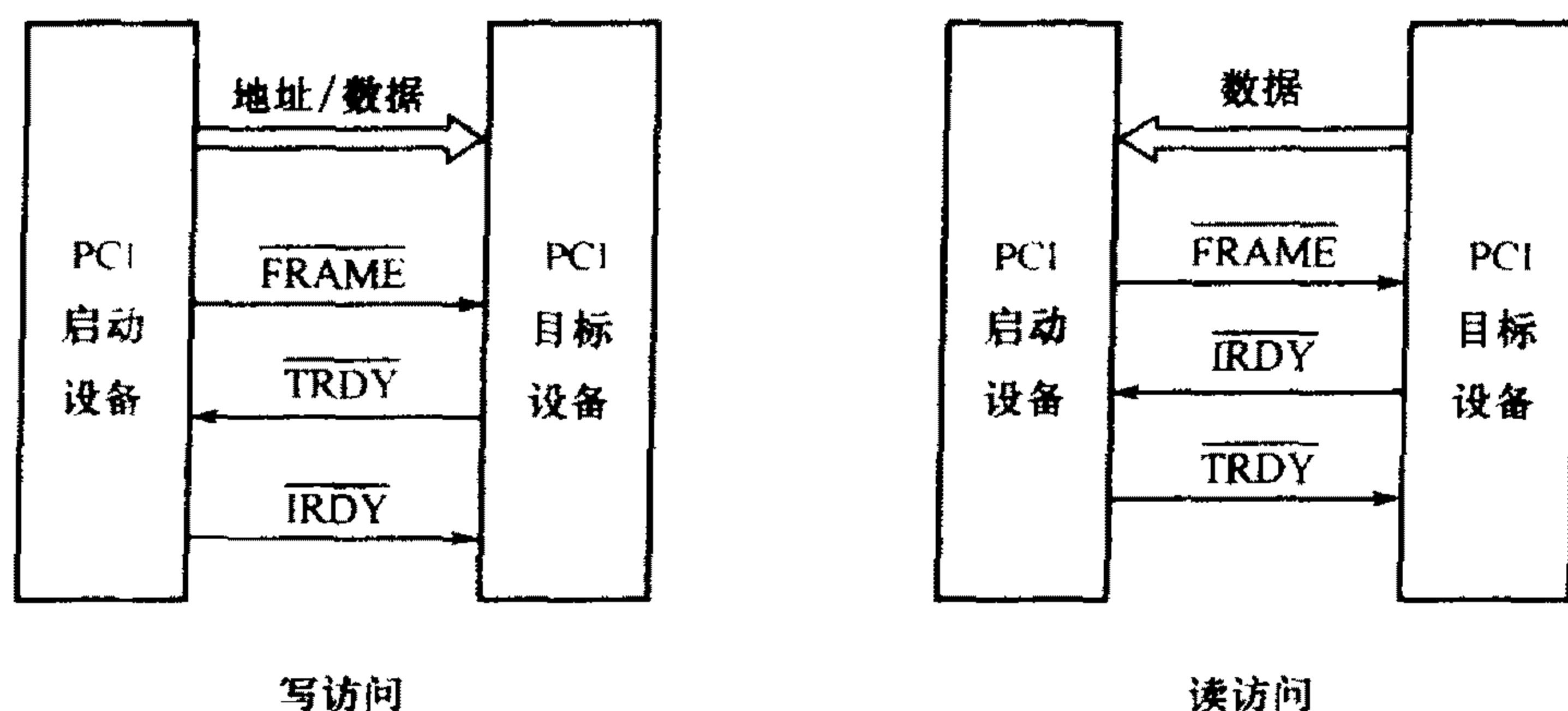


图 4.16 PCI 总线数据交换过程

6. PCI 总线的命令简介

PCI 总线命令主要有:

- ① INTA 序列 在中断响应序列期间,中断控制器被寻址和访问,以读取中断向量。在字节读操作期间,将读回一个字节的的中断向量。
- ② 特殊周期 用于把处理器状态的信息传输给所有的 PCI 设备。其低 16 位包含了 CPU 信息代码,如 0000H 表示处理器停止工作,0001H 表示处理器挂起,0002H 表示 80X86 的特殊编码或数据,0003H ~ FFFFH 为保留代码。
- ③ I/O 读 从指定的 I/O 地址(在引脚 $AD_0 \sim AD_{15}$ 上)端口读一个数据。I/O 读不支持突发操作。地址信号线译码后用于决定执行的是 8 位读还是 16 位读。
- ④ I/O 写 往指定的 I/O 地址(在引脚 $AD_0 \sim AD_{15}$ 上)端口写一个数据。
- ⑤ 存储器读 从连接到 PCI 总线上的存储器读出数据。命令/字节选通信号($C/\overline{BE}_3 \sim C/\overline{BE}_0$)用于决定访问的字节个数。
- ⑥ 存储器写 向连接到 PCI 总线上的存储器写入数据。命令/字节选通信号($C/\overline{BE}_3 \sim C/\overline{BE}_0$)用于决定访问的字节个数。
- ⑦ 读配置存储器 从 PCI 设备的配置存储器中读出配置信息。
- ⑧ 写配置存储器 把数据写入 PCI 设备的配置存储器,地址由读配置存储器命令指定。
- ⑨ 内存多路读 连续读指定地址开始的多个数据。直到 \overline{FRAME} 无效。
- ⑩ 双地址存取 在一个或两个时钟周期内传输 64 位地址到 PCI 设备(常只有 32 位是有用的)。如果是在一个时钟周期内完成,那么地址线 $AD_{63} \sim AD_0$ 包含 64 位地址(Pentium 处理器只有 32 位地址总线,但是这种模式可以用于支持其他系统的工作)。如果地址只能传输 32 位,那么首先把低 32 位地址放在 $AD_{31} \sim AD_0$ 信号线上,接着再传输高 32 位地址,总

共用两个时钟周期完成。

- 线性存储器读 类似于内存多路读。
- 带刷新的存储器写 用于执行多路数据写传输。此操作旁路了 Cache 的回写功能。

7. PCI 总线的中断和总线仲裁

PCI 总线支持四种中断(\overline{INTA} 、 \overline{INTB} 、 \overline{INTC} 、 \overline{INTD})。任何 PCI 设备接口都可以使用 \overline{INTA} ，但是只有多功能接口可以使用其他三种中断(\overline{INTB} 、 \overline{INTC} 、 \overline{INTD})。根据系统 BIOS 的设置，这些中断可以被 PCI 桥接器导入 IRQ_x 中断中的一个。例如，100 Mb/s 的以太网卡可以触发中断 \overline{INTA} ，该中断将被导入 IRQ_{10} 。

在 PCI 总线上，总线主控器是总线上的设备，它可以接管总线的控制权。为了避免总线冲突，PCI 使用 \overline{REQ} (请求)信号和 \overline{GNT} (授权)信号来对总线的使用进行仲裁。当某一总线主控设备需要控制总线时，它就会激活 \overline{REQ} 信号，用以表示请求控制 PCI 总线，若请求被接受，仲裁逻辑电路就激活 \overline{GNT} 信号，于是发出请求信号的总线主控设备就可以获得总线控制权。

8. PCI 接口的配置寄存器

每个 PCI 接口都有 256 B 的配置存储器，每 4 B 构成一个 32 位的寄存器，所以配置存储器可以看成是由 64 个 32 位的寄存器组成。配置存储器的前 64 B 为预定义信息区(又称标题区)，其中的信息由 PCI SIG 组织进行预定义；后 192 B 为特殊配置数据区，其中的内容根据接口的特性由生产厂家定义。主机系统可以通过配置存储器给出的信息来配置操作系统，以实现 PCI 接口的即插即用特性。图 4.17 给出了配置存储器的前 64 B 的内容。下面简单介绍其中主要字段的含义。

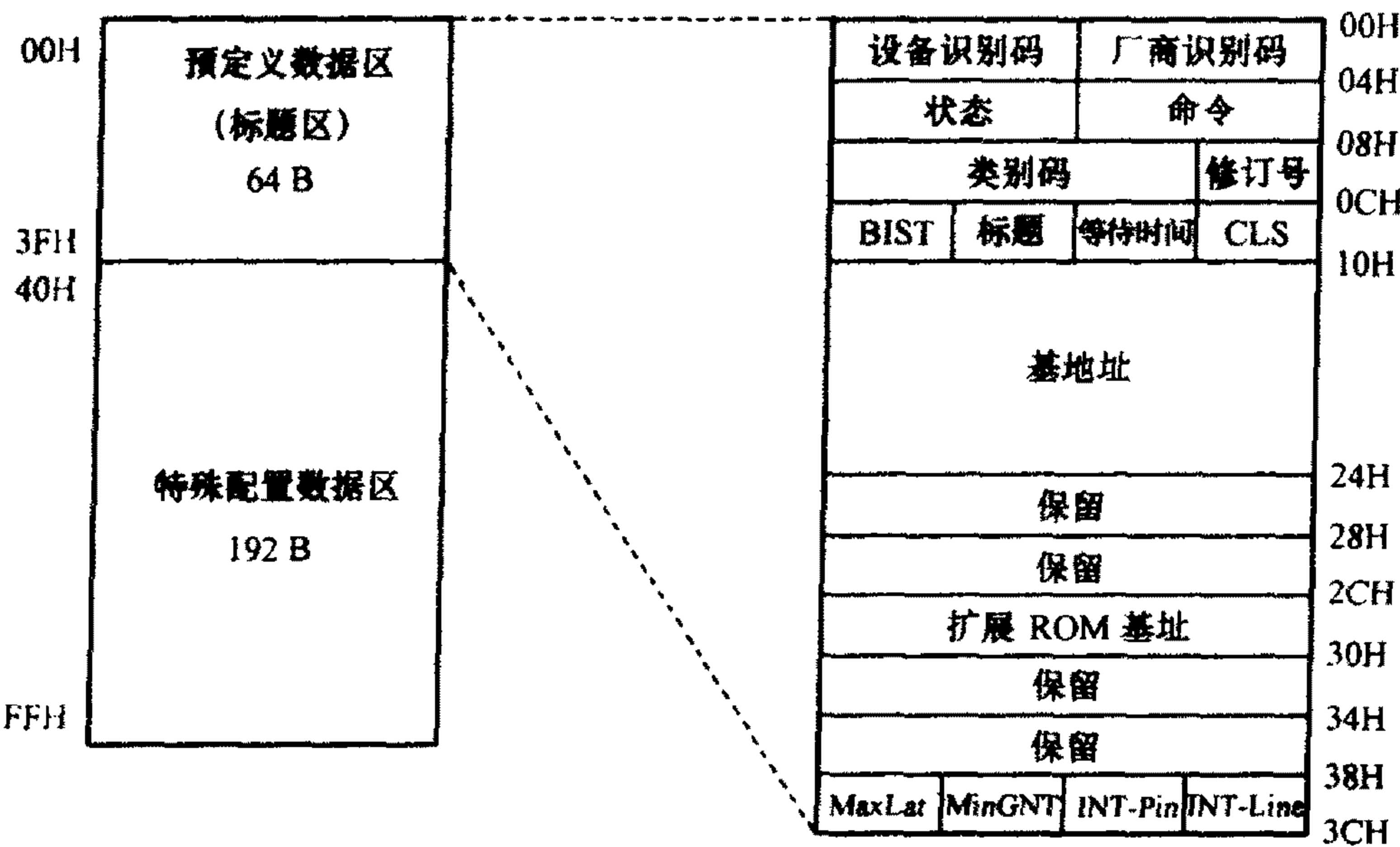


图 4.17 PCI 接口卡上的配置存储器格式

① 设备识别码和厂商识别码 设备识别码若为 FFFFH,表示没有安装设备,而其他数据都表示设备的编码。厂商识别码是由 PCI SIG 组织进行分配的。这个编码一般显示在 BOIS 的启动画面里,例如, Motorola 的厂商识别码为 1057H, EPSON 为 1008H, Intel 为 8086H, 等等。

② 状态和命令 格式如图 4.18 所示。

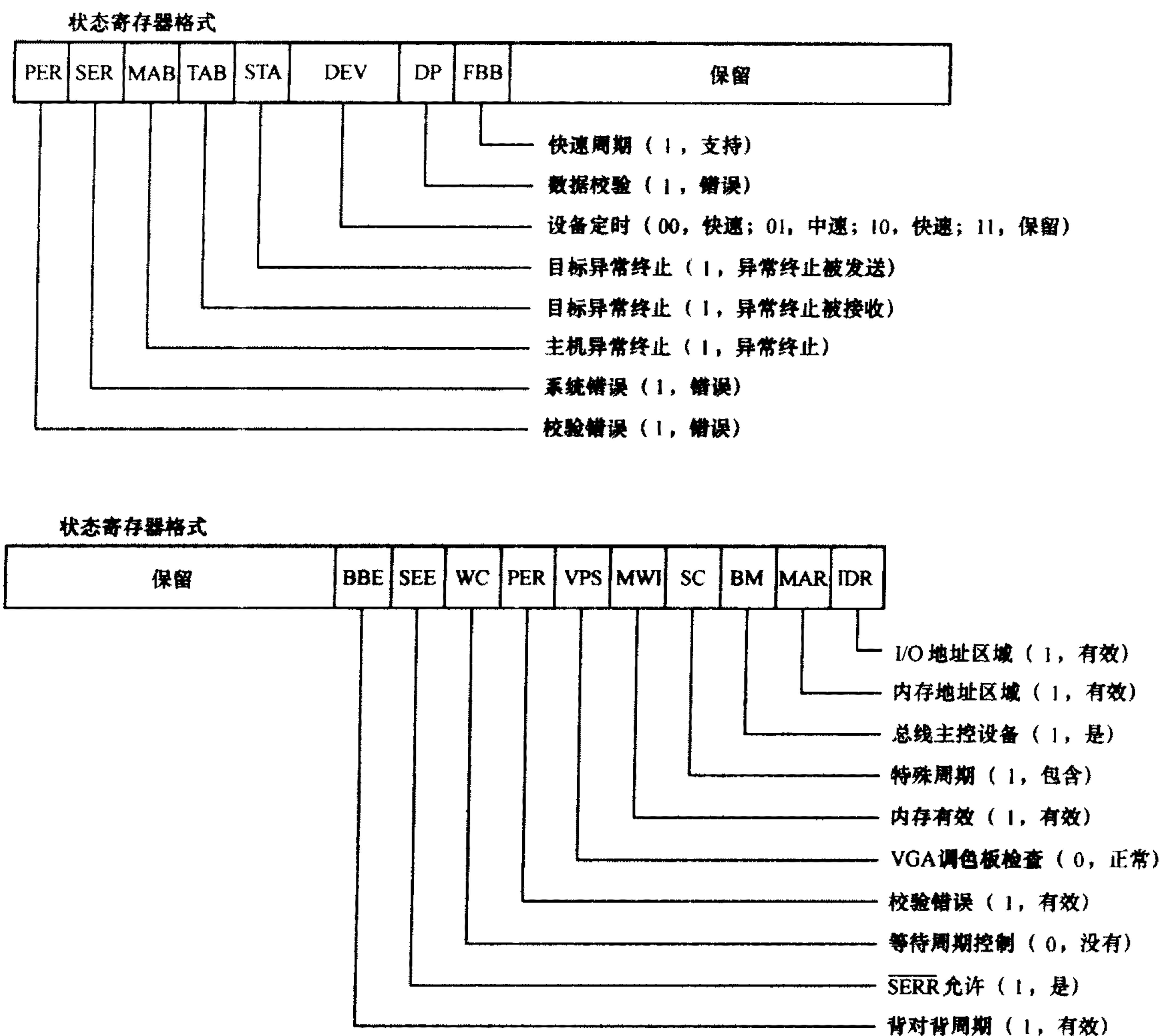


图 4.18 PCI 接口卡上配置存储器中状态和命令的格式

③ 类别码和修订号 类别码有 3 个字节,第一个字节定义部件的主类别(00H 表示没有类码,01H 表示大容量存储器,02H 表示网络控制器,03H 表示视频控制器,04H 表示多媒体组件,05H 表示内存控制器,06H 表示桥接器),第二个字节为设备的子类码。第三个字节

用于定义组件的编程接口。前两个字节的代码定义见表 4-3。

表 4-3 PCI 设备类别码

类别码	设备	类别码	设备
0000H	早期的非 VGA 设备(非 PnP)	0401H	音频多媒体
0001H	早期的 VGA 设备(非 PnP)	0480H	其他多媒体控制器
0100H	SCSI 控制器	0500H	RAM 控制器
0101H	IDE 控制器	0501H	FLASH RAM 控制器
0102H	软盘控制器	0580H	其他存储器控制器
0103H	IPI 控制器	0600H	主机桥
0180H	其他硬盘/软盘控制器	0601H	ISA 桥
0200H	以太网控制器	0602H	EISA 桥
0201H	令牌网控制器	0603H	MCA 桥
0202H	FDDI 控制器	0604H	PCI - PCI 桥
0280H	其他网络控制器	0605H	PCMCIA 桥
0300H	VGA 控制器	0680H	其他桥
0301H	XGA 控制器	0700H ~ FFFE H	保留
0380H	其他视频控制器	FFFFH	其他类别的设备
0400H	视频多媒体		

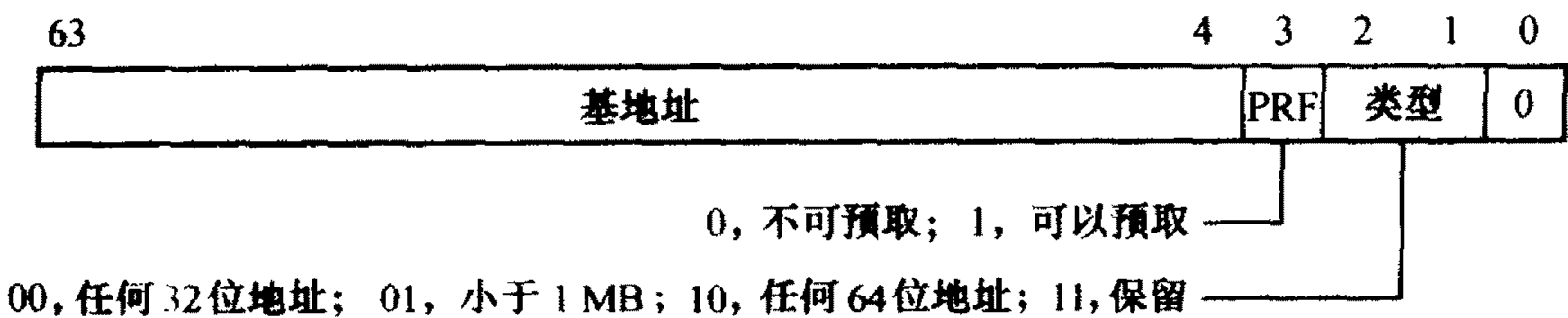
④ BIST(内部自测) 一个 8 位字段,其中最高有效位定义了设备是否可以进行一次 BIST,接下来的次高有效位定义了是否将要执行一次 BIST(该位置上的 1 表示要执行一次 BIST)。第 0~3 位定义了 BIST 执行过后的状态码(0 值表示没有错误)。

⑤ 标题字段 定义了在最开始 16 B 的报头和以后的 48 B 的布局,其最高有效位定义了该设备是否是多功能设备,1 表示是多功能设备。

⑥ 等待时间 定义了 PCI 总线操作所需的时间,其中的时间值为等待时间加上 8 个 PCI 时钟周期。

⑦ CLS(高速缓存行大小)字段 定义了部件中的高速缓存行的大小。

⑧ 基地址 该区域存放了内存的基地址、I/O 地址空间和扩展 ROM 的地址。前两个字节包含位于 PCI 接口中内存的 32 位基地址或 64 位基地址。内存基地址的格式为:



第三个双字包含 I/O 空间的基地址,其格式如下:

63	32 31	2	1	0
保留		基地址		0 1

注意,尽管 Intel 微处理器只使用 16 位的 I/O 地址,但却为将来扩充到 32 位 I/O 地址预留了空间。32 位 I/O 地址目前可用于 680X0 系列微处理器和 Power 微处理器系统中。

⑨ 扩展 ROM 基址 允许将扩展 ROM 的地址放到 32 位存储器地址区域里的任何位置。

⑩ MaxLat 和 MinGNT 只读字段 定义了最大和最小等待时间值。

⑪ INT - Line 一个只有 4 位的字段,用于定义所用到的中断信号线(IRQ₀ ~ IRQ₁₅),字段值 0000 ~ 1111 分别表示使用 IRQ₀ ~ IRQ₁₅。PCI 桥接器可以将这个中断转接到适当的 IRQ 信号线上。

⑫ INT - Pin 一个 4 位的字段,它定义了设备正在使用哪一根 PCI 中断信号线($\overline{\text{INTA}}$ ~ $\overline{\text{INTD}}$),0 表示没有中断,1 表示使用 $\overline{\text{INTA}}$,2 表示 $\overline{\text{INTB}}$,依此类推。

9. I/O 寻址

标准 PC 的 I/O 寻址范围是 0000H ~ FFFFH 的 64 KB 空间,只需要 16 位地址线,而 PCI 总线可以支持 32 位或 64 位寻址。PCI 设备可以按以下两种方式进行配置:

1) 配置方式 1

通过两个标准地址(0CF8H 和 0CFCH)来配置 PCI 设备,见表 4-4。其中地址为 0CF8H 的 4 个字节称为配置地址寄存器;0CFCH 的 4 个字节称为配置数据寄存器。

表 4-4 地址配置说明

地 址	名 称	说 明
0CF8H	配置地址寄存器	用于访问配置地址区域
0CFCH	配置数据寄存器	用于从 PCI 设备的配置存储器中读一个 32 位数或把一个 32 位数写入 PCI 设备的配置存储器中

配置地址寄存器的格式如图 4.19 所示。

2) 配置方式 2

这种方式把每个 PCI 设备映射到 C000H ~ CFFFH 的 4 KB 大小的 I/O 空间中。要使用配置方式 2,应该激活 0CF8H 地址处的配置寄存器 CSE。CSE 寄存器的格式定义如下:

b₇ ~ b₄ 为 0000 时为正常模式,为 0001 ~ 1111 时激活配置区。也就是说,这 4 位的值除了 0000 以外,其他任何数值都可以将映射配置区激活。激活后,C000H ~ CFFFH 之间的所有

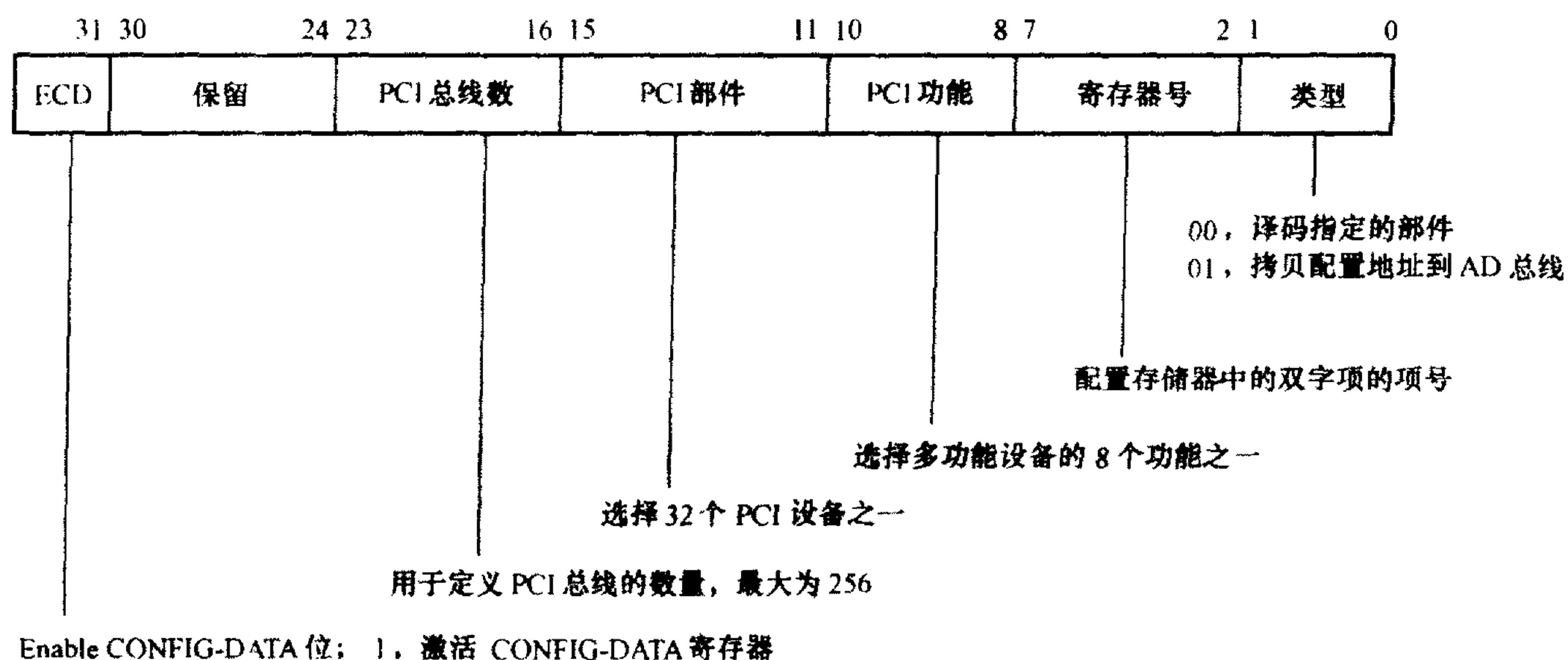


图 4.19 配置地址寄存器的格式

I/O 地址都可以当成是正常的 I/O 端口来操作。

$b_3 \sim b_1$ 为功能位。定义了多功能 PCI 设备中的功能数。

b_0 为 SCE 位。0 表示配置周期,1 表示特殊周期。

10. PCI 总线的发展

当前 PCI 总线的最高版本是 2.1 版,虽然在理论上达到 66 MHz 的时钟频率,数据传输速率可达 533 MB/s,但对于新型的 CPU(如 Xeon、Katmai、K7 等外频可达 100/133 MHz,而 Pentium 4 的前端总线频率已达 533 MHz)、高总线频率主板以及各种新型外设对传输带宽日益增长的需求,PCI 总线已经不堪重负。例如,一块 32 位的千兆位以太网卡(125 MB/s)和一块 IEEE 1394 接口卡(50 MB/s)所需的传输带宽即可占满 PCI 总线的全部带宽。

为了满足系统对总线带宽的需求,Intel 推出的新一代 PCI 总线规范称为 PCI-X,主要适用于 133 MHz 总线时钟频率的台式机主板。更新型的 PCI-X 2.0 可适用于总线时钟频率为 533 MHz 的新型主板。PCI-X 与传统 PCI 总线的比较见表 4-5。

表 4-5 PCI-X 总线与传统 PCI 总线的比较

类 型	PCI-32	PCI-64	PCI-X	PCI-X 2.0
支持外设数量	6(共享)	6(共享)	4	未发布
总线时钟频率/MHz	33	33/66	66/100/133	266/533
数据传输速率/MB/s	133	266、533	533、800、1066	2100/4200

续表

类 型	PCI - 32	PCI - 64	PCI - X	PCI - X 2.0
时钟同步方式	与 CPU 及时钟频率有关	与 CPU 及时钟频率有关	与 CPU 及时钟频率无关	与 CPU 及时钟频率无关
总线位宽/b	32	64	64	64
工作电压	3.3 V/5 V	3.3 V/5 V	3.3 V	3.3 V
引线脚数	84	120	150	未发布

除此之外, Intel 还准备推出一种称为 Mini PCI 的总线标准。Mini PCI 对原来的 PCI 总线在控制线路和功能上做了改进, 减小了外形尺寸, 使之适用于便携式计算机。

4.4.4 AGP 总线

1. 为什么要采用 AGP

AGP 是一种专为提高视频带宽而设计的总线规范。AGP 插槽可以插入符合该规范的 AGP 显示插卡。其视频信号的传输速率可以从 PCI 的 133 MB/s 提高到 266 MB/s(×1 模式)、533 MB/s(×2 模式)、1 066 MB/s(×4 模式)或 2 133 MB/s(×8 模式)。严格地说, AGP 不能称为总线, 因为它仅在 AGP 控制芯片和 AGP 显示卡之间提供了点对点的连接。

在 AGP 出现以前, 几乎所有图形显示卡都采用了 PCI 总线接口。随着图形显示卡 3D 图形处理性能的提高, 显卡处理的数据越来越多, PCI 接口就逐渐地暴露出它的局限性。这种局限性主要表现在 3D 图形描绘中, 储存在 PCI 显示卡显示内存中的不仅有影像数据, 还有纹理数据(Texture Data)、Z 轴的距离数据及 Alpha 变换数据等, 特别是纹理数据的信息量相当大。如果要描绘细致的 3D 图形, 就要求显存容量很大, 而且必须采用快速的显存, 以至于造成显示卡价格高昂。为此, 3D 显示卡的制造厂商所期望的是既能增加纹理数据的储存能力, 又能降低产品的成本。一个有效的办法就是将纹理数据从显示内存移到主内存, 以便减少显示内存的容量, 从而降低显示卡的成本。从整个系统来看, 增加显示内存也不如增加主内存划算, 因为用作主存储器的 DRAM 的价格已不太昂贵, 而且把纹理数据储存在主存储器比储存在显示存储器更可有效利用内存。存储纹理数据所需的内存空间依应用程序而定, 也就是说, 当应用程序结束后, 它所占用的主内存空间又可恢复, 纹理数据不会永远占用主内存的空间。

然而遗憾的是, 当纹理数据从显示内存移到主内存时, 由于纹理数据传输量很大, 数据传输的瓶颈就从显示卡上的内存总线转移到了 PCI 总线上。例如, 显示 $1\,024 \times 768 \times 16$ 位真彩色的 3D 图形时, 纹理数据的传输速度需要 200 MB/s 以上, 但目前的 PCI 总线最高数据传输速度仅为 133 MB/s, 因而成为系统的主要瓶颈。3D 绘图时所需数据传输带宽见表 4-6。

表 4-6 3D 绘图时所需数据传输带宽

扫描像素带宽/(MB·s ⁻¹)	640 × 480 像素	800 × 600 像素	1024 × 768 像素
显示器输出	50	100	150
显示内存刷新	100	150	200
Z 缓冲存取	100	150	200
纹理存取	100	150	200
其 他	20	30	40
合 计	370	580	840

为了解决 3D 图形数据的传输问题,主要的微型计算机生产厂商联合推出了 AGP 图形接口。AGP 在主内存与显示卡之间提供了一条直接的通道。使得 3D 图形数据可以不经过 PCI 总线,而直接送入显示子系统。这样就能突破由于 PCI 总线形成的系统瓶颈,从而实现了以相对低价格来达到高性能 3D 图形的描绘功能。所以推出 AGP 接口的主要目的就是为大幅度提高微型计算机的 3D 图形的处理能力,或者说 AGP 是用于加速图形显示的一个专用总线接口。采用 AGP 总线的系统结构如图 4.20 所示。

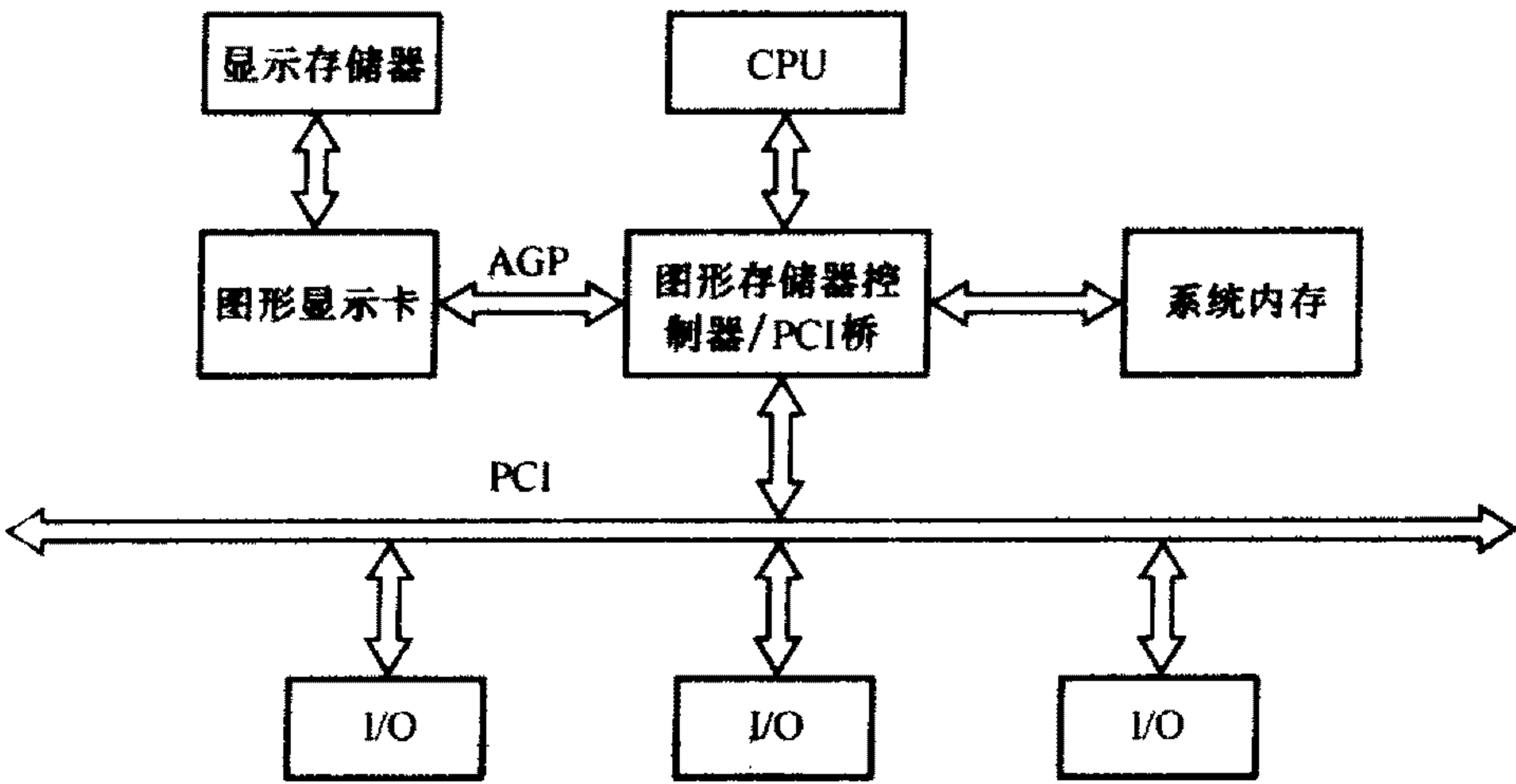


图 4.20 AGP 接口的系统结构

2. AGP 的性能特点

AGP 以 66 MHz PCI Rev2.1 规范为基础,在此基础上扩充其主要功能。

1) 数据读/写操作的流水线操作

流水线(Pipelining)操作是 AGP 提供的仅针对主存的增强协议。由于采用了流水线操作减少了内存等待时间,数据传输速度有了很大提高。

2) 具有 2X、4X、8X 的数据传输频率

AGP 使用了 32 位数据总线和多时钟技术的 66 MHz 时钟。因为时钟频率提高到了 66 MHz 所以带宽是 PCI 总线的两倍,达到了 266 Mb/s。随后很快 AGP 2X 问世,通过每周期传送两次 32 位数据将带宽提高到了 533 Mb/s。以后又出现了每时钟周期处理 4 个 32 位数据的 AGP 4X 模式,使 AGP 总线传输带宽突破了 1 Gb/s,达到了 1 066 MB/s。最新的 8X 模式使 AGP 带宽达 2 133 MB/s。

3) 直接内存执行 DIME

AGP 允许 3D 纹理数据不存入拥挤的帧缓冲区(即图形控制器内存),而将其存入系统内存,从而让出帧缓冲区和带宽供其他功能使用。这种允许显示卡直接操作主存的技术称为 DIME(Direct Memory Execute)。应该说明的是,虽然 AGP 把纹理数据存入主存,也可以称为 UMA(Unified Memory Architecture,统一内存体系结构)技术,但是与一些低端机采用的 UMA 有以下两点区别:

- 通过 AGP 技术使用的主内存(称为 AGP RAM)并没有完全取代显示卡的显示缓存,AGP 主存只是对缓存的扩大和补充。
- 低端机的 UMA 是通过 PCI 接口运行的,其速度较慢。

4) 地址信号与数据信号分离

采用多路信号分离技术(Demultiplexing),并通过使用边带寻址 SBA(Side Band Address)总线来提高随机内存访问的速度。

5) 并行操作

在 CPU 访问系统 RAM 的同时允许 AGP 显示卡访问 AGP 内存,显示卡可以独享 AGP 总线带宽,从而进一步提高了系统性能。

3. AGP 的工作模式

AGP 的工作模式如表 4-7 所示。

表 4-7 AGP 的工作模式

版本	模式	工作频率	数据传输率/MHz	传输方式/(MB·s ⁻¹)	额顶工作电压/V
1.0,2.0	1X	66	266	上升沿	3.3
1.0,2.0	2X	133	533	上升沿和下降沿	3.3
2.0,3.0	4X	266	1 066	上升沿和下降沿	1.5
3.0	8X	533	2 133	上升沿和下降沿	1.5

从上表中可以看出,要真正达到良好的 3D 图形处理能力,应该采用 2X 以上的工作模式。在 1X 模式下,由于带宽不足,并不能适合 DIME 的速度,3D 图形处理能力仍然是不理想的。因此在选购主板和 AGP 显示卡时,要注意它们是否支持 AGP 2X、4X 或 8X 的工作模

式。要注意的是,最新的 AGP 8X 模式在目前的大多数显卡上并不能显著提高性能,因为显卡的数据处理能力还远小于 AGP 8X 模式能够提供的能力,只有在显卡性能进一步提升后,AGP 8X 模式才能表现出它的巨大威力。

另外,AGP 1X 和 AGP 2X 的工作电压为 3.3 V,而 AGP 4X 和 AGP 8X 的工作电压仅为 1.5 V,所以在使用时一定不要把 AGP 1X 和 AGP 2X 的显卡插入 AGP 4X 或 AGP 8X 插槽中,否则有可能烧毁主板。

5. PCI 和 AGP 的比较

表 4-8 列出了台式机中 PCI 和 AGP 的性能比较。

表 4-8 PCI 和 AGP 的性能比较

性 能	PCI - 32	AGP
传输方式	同步	同步
内存优先存取	不支持	不支持
数据线位宽	32 位	32 位
总线时钟/MHz	33	66
传输带宽/(MB·s ⁻¹)	133	2166(8X)
支持的插槽个数	≤5	1

在采用 AGP 的系统中,由于显示卡通过 AGP、芯片组与主内存相连,提高了显示芯片与主内存间的数据传输速度,让原需存入显示内存的纹理数据,现可直接存入主内存,这样可提高主内存的内存总线使用效率,也提高了画面的更新速度及 Z Buffer(Z 缓冲)等数据的传输速度,而且还减轻了 PCI 总线的负载,有利于其他 PCI 设备充分发挥性能。在 PC98 规格中,ISA 总线已被取消,ISA 设备终将被淘汰,所以,把占用了 PCI 总线大量带宽的显示卡移到 AGP 上是非常必要的。当然 AGP 不可能取代 PCI,因为我们已经知道 AGP 仅仅是一个图形显示接口,而不是系统总线。

AGP 插槽和 AGP 插卡的引脚都采用了与 EISA 相似的上下两层结构,因此减小了 AGP 插槽的尺寸。图 4.21 中灰色的插槽为 AGP 插槽的外形图。

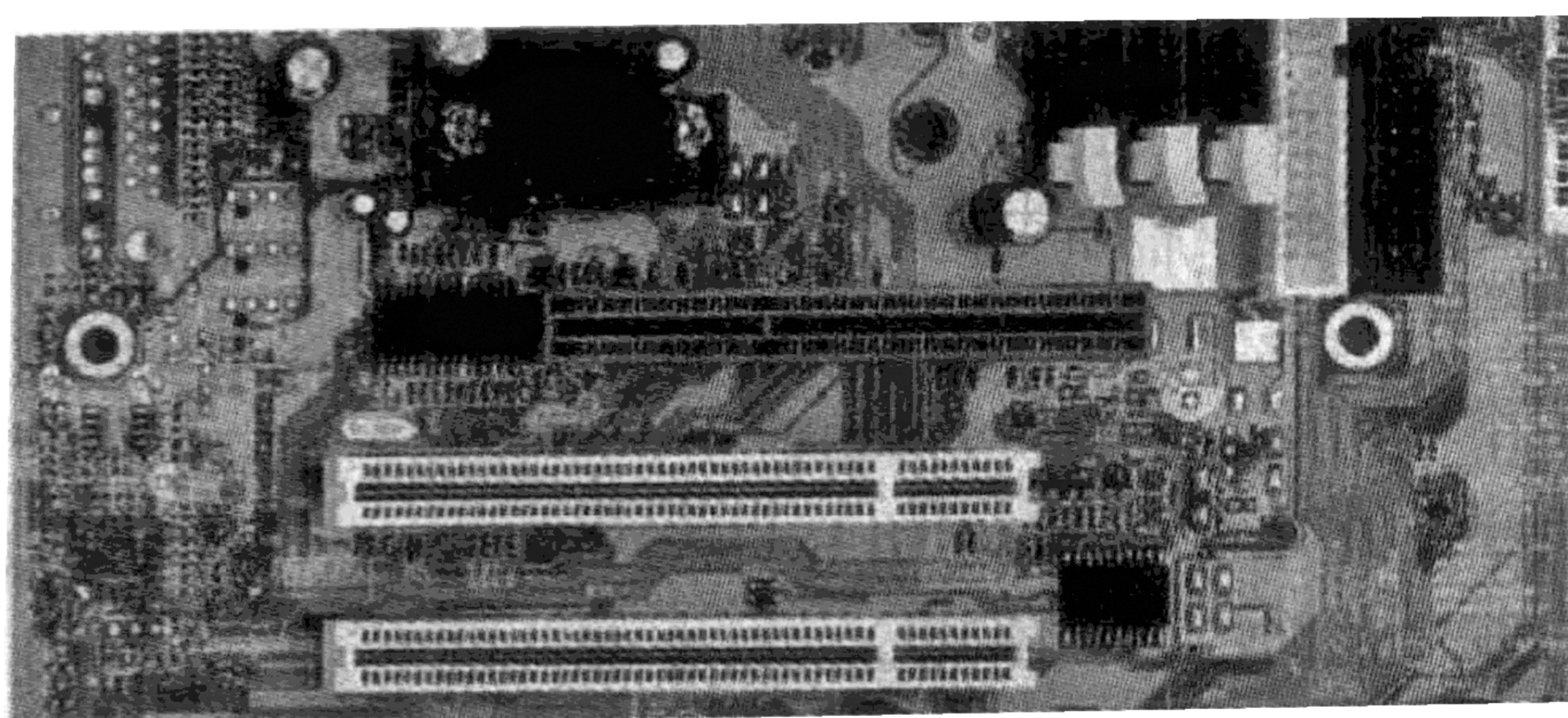


图 4.21 AGP 插槽外形图

4.4.5 新型总线 PCI Express

1. I/O 总线的瓶颈

随着大规模集成电路技术的不断进步,微处理器性能的提高已经到了令人惊讶的程度,目前微处理器的工作频率已经高达 3 GHz,预计在 3~5 年,用户就可以用上 10 GHz 的微处理器。当人们每一次在为处理器性能提高而欢呼的时候,同时又悲哀地发现计算机的整体性能并没有随着微处理器性能的提高而同步增长。现在人们已经意识到,高速微处理器所能发挥的性能与其所在系统的其他组件密切相关。保持各组件在整体上处于同等水平,能够以最佳状态协作而不受性能低下的组件及技术的拖累是提高微型计算机系统性能的关键所在。

分析近十年来微型计算机结构的发展,可以看出,I/O 子系统在微型计算机系统中是发展最不平衡的组件之一。作为 I/O 系统的高速公路,PCI 总线自从进驻微型计算机以来,10 年中几乎没有什么大的改进。早期的 PCI 总线是为了彻底解决 ISA 总线速度慢、难于管理的弊端而诞生的。到后来,作为一个开放标准,PCI 已从微型计算机通用 I/O 互连通路的最初角色,演变成为在桌面 PC、服务器、工业控制和通信平台中无所不在的 I/O 接口标准。

虽然 PCI 的优点很多,但多媒体技术的出现使得 PCI 的局限性逐渐显现出来。用户往往要求微型计算机能够提供更强大的多媒体能力,例如,在当前的主流多媒体 PC 上,用户往往同时具有声卡、网卡、3D 图形加速卡、视频采集卡及 IEEE 1394 接口卡等设备。这些设备工作时将产生极大的数据流量。例如,理想状况下一块 32 b 的千兆位以太网卡在满负荷工作时的数据流量为 125 MB/s,IEEE 1394 接口卡的数据流量为 50 MB/s 等。而台式微型计算机中 PCI 总线所能提供的最大带宽为

$$33 \text{ MHz(额定工作频率)} \times 32 \text{ b(总线位宽)} = 1\,066 \text{ Mb/s} = 133 \text{ MB/s}$$

显然,仅千兆位网卡产生的数据便足以独占整个 PCI 总线的带宽。这便使得 PCI 总线面临一个非常尴尬的处境——3D 图形加速卡、千兆位以太网卡、IEEE 1394、移动对接设备及其他附件的飞速发展,以及它们所需要的更大带宽已经使 PCI 总线不堪重负,再也无法及时处理这些设备所发送的并发/多路数据流,它已逐渐成为当前微型计算机性能的瓶颈。

为了解决这个问题,人们开始把一些数据流量非常大的 I/O 工作从 PCI 中剥离出来,由一个专用接口来负责,例如前面介绍过的用于 3D 图形加速卡的 AGP 接口就是一个典型的例子。此外,芯片组的 HUB Link、V-Link 等技术也使得南北桥芯片之间的连接脱离了 PCI 总线规范的控制。然而这些改变都只是局部的,真正要彻底解决 PCI 的瓶颈效应,必须从根本上改变总线设计,采用一种新的总线来彻底取代 PCI。

由 Intel 等开发的 PCI Express(原名 3GIO,第 3 代 I/O 总线)就是为满足这一需求而推出

一种新型的高速串行 I/O 互连接口。

2. PCI Express 概览

PCI Express 是一种串行总线,其最大数据传输速率为 8.2 GB/s。与 PCI 相比,PCI Express 的导线数量减少了将近 75%,但它提供的速度却几乎达到 PCI-X 2.0 的两倍,而且很容易进行扩充。在 PCI 总线上,PCI 对所有设备都是平等对待的,所有设备共享同一总线资源,而 PCI Express 采用的是点对点技术,它能够为每一个设备分配独享的通道,彻底消除了设备之间由于共享资源带来的总线竞争现象。按照 PCI Express 规范,每个设备最多可以通过最多 64 根 PCI Express 连接线和设备建立连接,这 64 根线中每根的传输速率约为 26 MB/s,每个连接占用的带宽可在 1 根、2 根、4 根、8 根、16 根或 32 根连接线之间进行定义,以实现更高的集合速度。例如,用户可以使用单线连接得到 206 Mb/s 的传输速度,使用 8 线连接得到 1.66 Gb/s 的传输速度,使用 32 线连接得到 6.6 Gb/s 的传输速度等。通过增添更多通道可以轻松扩展带宽。

PCI Express 在点对点架构基础上为高速接入设备提供了一种全新的控制单元——交换器,如图 4.22 所示。交换器的作用主要是对高速 PCI Express 设备之间的点对点通信进行管理和控制。举例来说,如果一块网卡发出的数据需要传送给另一块网卡,那么数据通过交换器可以直接在两块网卡之间传递,不需要再通过系统 I/O 桥进行交换,从而节省了系统 I/O 传输带宽。该技术与 DMA 相比更具优势,可以保证与系统其他部分之间良好的透明性,系统内存和处理器可以自由执行其他操作而不受任何影响。

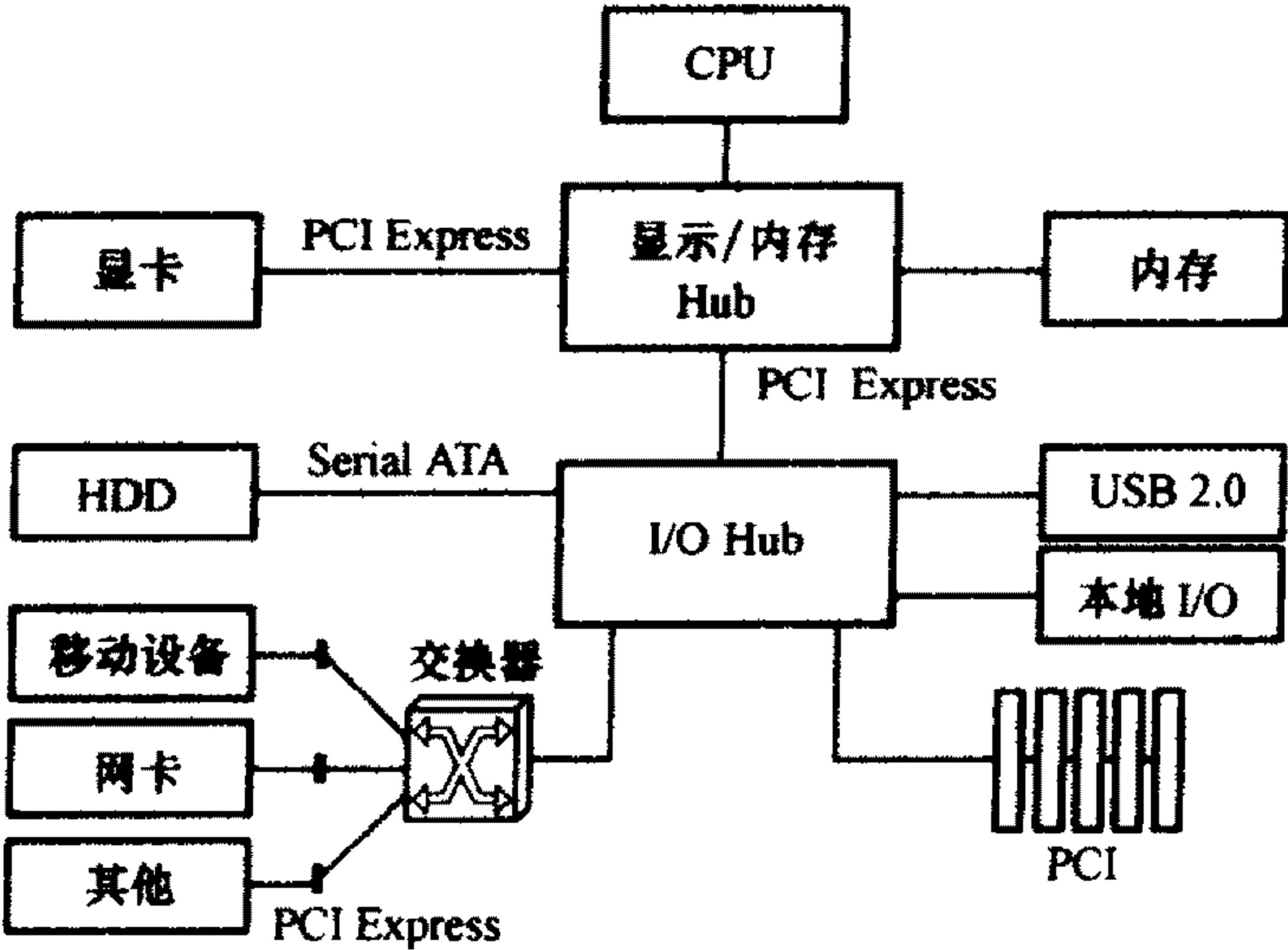


图 4.22 PCI Express 结构

PCI Express 的主要技术指标见表 4-9 所示。

表 4-9 PCI Express 的主要技术指标以及与 PCI 的比较

	PCI-32	PCI-X 1.0	PCI Express
支持外设数量	6	4	64(单线)
总线时钟频率/MHz	33	66/100/133	2500
最大数据传输速率/(MB·s ⁻¹)	133	1066	8200
时钟同步方式	与 CPU 及时钟频率有关	与 CPU 及时钟频率无关	内建时钟
总线位宽	32 位并行	64 位并行	串行
工作电压	3.3 V/5 V	3.3 V	未发布
引线脚数	84	150	40

3. PCI Express 的系统结构

PCI Express 的系统结构采用了和 OSI 网络模型相类似的分层模型,不过 PCI Express 只有 5 层,而不是 OSI 的 7 层。PCI Express 兼容 PCI 寻址模型,确保了它能够在无需作任何更动的前提下继续支持现有的应用程序和驱动程序。图 4.23 所示为 PCI Express 的分层结构。

PCI Express 的分层结构模型自上而下由软件层、会话层、事务处理层、数据链路层和物理层组成。其配置采用在 PCI 即插即用规范中定义的标准机制。软件层产生的读/写请求被事务处理层采用数据包封装协议传送给各种 I/O 设备。数据链路层为这些数据包增加顺序号和 CRC 校验码以实现高度可靠的数据传输机制,为高层提供一个无差错的数据传输链路。物理层实现数据编/解码和多个通道数据拆分/解拆分操作,每通道都是全双工的,可提供 2.5 Gb/s(200 MB/s)的传输速率。

目前,PCI Express 还正在开发中,它要得到广泛的认可需要各行业的支持。PCI Express 主要应用领域包括台式机、服务器、通信和嵌入式应用。预计图形显示卡和 10 GB 以太网接口卡将率先采用 PCI Express,并于 2003 年推出。

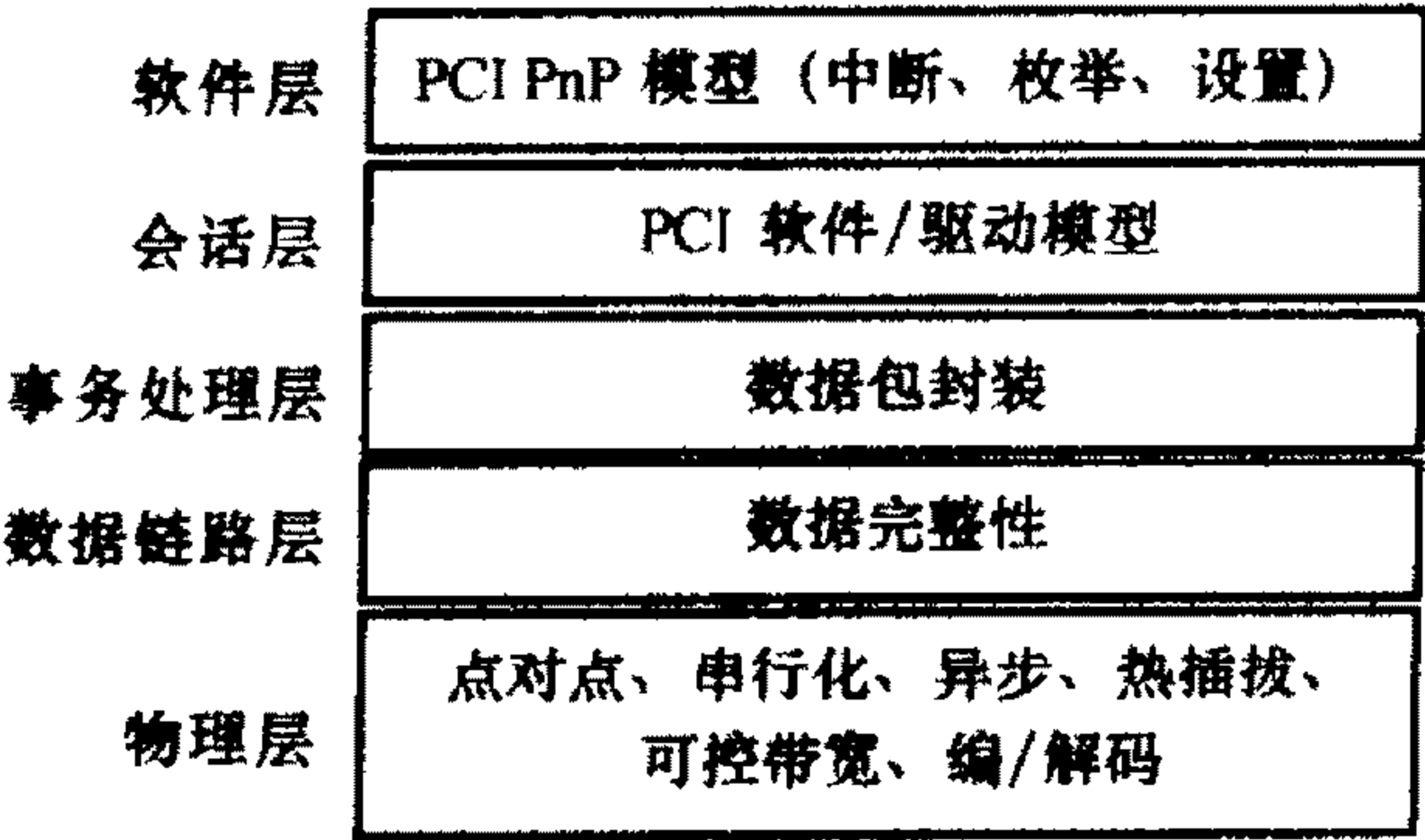


图 4.23 PCI Express 的分层结构模型

4.5 外部设备总线

外部设备总线用于实现计算机主机和外部设备之间的连接,它与传统外设接口有很大的区别,传统外设接口是专用的,通常只能连接某一特定类型的设备,而且大多数情况下只

能连接一个设备。外部设备总线是通用的,可连接不同的外部设备,并且允许在一个总线上连接多个设备。

常见的外部设备总线有 USB(Universal Serial Bus)和 IEEE 1394(又称 FireWire)。

4.5.1 通用串行总线(USB)

传统的计算机仅有少量 SIO 和 PIO 接口,通常设置在主机箱的后面板上,用以连接键盘、鼠标器、显示器与打印机等常用外部设备。随着计算机的应用日益广泛,需要连接的外设不断增加,接口短缺的矛盾日趋尖锐,于是 USB 接口和 USB 总线应运而生。

1. USB 概述

USB 是由 Compaq、DEC、IBM、Intel、Microsoft 和 NEC 等多家公司共同开发的一种新的外设连接技术。早在 1995 年,这些公司就成立了一个称为通用串行总线应用论坛(Universal Serial Bus Implementers Found, USB-IF)的组织,旨在促进 PC 总线的标准化,加速新标准的制订和产品开发。现在参加该组织的成员已超过 400 家。该组织的目标是发展一种兼容低速和高速的技术,从而可以为广大用户提供一种可共享的、可扩充的、使用方便的串行总线。该总线应独立于主计算机系统,并在整个计算机系统结构中保持一致。为了实现上述的目标,USB-IF 发布了一种称为通用串行总线的串行技术规范,简称为 USB。由于微软从 Windows 98 开始加入了对 USB 的支持,使 USB 技术得到了飞速发展和极为广泛的普及。现在,USB 已成为微型计算机上普遍认同的一种事实上的接口标准,支持这一标准的各种新产品正在大量涌现。

USB 的推出来源于 4 个方面的考虑:

1) 提供 PC 与电话的连接

现在人们可以理解计算技术和通信技术的结合将是下一代计算机应用的基础。面向机器和面向人类的各类数据从一处位置或环境转移到另一处的运动有赖于到处存在的且十分廉价的连接。不幸的是,计算机工业和通信工业在以往都是独自发展的,它们尚不能提供便宜、方便、即插即用的连接手段。而通用串行总线则可提供无处不在的各式各样的 PC 到电话的互相连接。

2) 方便使用

PC 在体系结构上缺乏灵活性已被公认为致命的弱点,严重影响到其应用和发展。虽然用户的友好图型界面和硬件的结合,以及新一代总线如 PCI、PnP ISA 及 PCMCIA 和相关软件的发展使计算机更容易掌握和使用。但从普通用户观点出发,PC 机的 I/O 接口例如串/并接口、键盘/鼠标/游戏杆接口等尚不具备即插即用的属性。

3) 接口扩充

目前,外设的增加继续受到可用端口的限制。实际上缺乏双向、廉价的、低到中速的外设总线已经阻碍了新一代多功能外设如电话/传真/调制解调器、答录电话、扫描仪、PDA、键盘、鼠标、便携式移动存储设备等的发展。现存的互连方法只是在某方面对产品进行了优化。随着每个新的功能或外设加到 PC 上,同时就增加了新的接口访问方式。

4) 可作为中低速设备总线

PC 接口中统治了很长时间的接口是串行接口和并行接口。它们有两个致命的弱点,一是传输速度低(串口最高为 115 kb/s, 并口最高为 1.5 Mb/s),二是连接设备少(一个接口只能连接一个设备)。

USB 接口较好地解决了以上这些问题。首先,USB 可以实现即插即用功能,允许外设热插拔,而不必关闭主机电源。USB 可智能识别 USB 链上外围设备的插入或卸出,这就为 PC 的外设扩充提供了一个很好的解决方案。第二,USB 采用“级联”方式,即每个 USB 设备可以连接到上一级设备的 USB 插座上,其本身又提供一个 USB 插座供下一个 USB 设备连接用。通过这种类似菊花链式的连接(星型结构),一个 USB 控制器可以连接多达 127 个外设。第三,每个 USB 外设间距离(线缆长度)可达 5 m,主机和 USB 设备之间还可利用 HUB 级联的方法延长连接距离,但最多允许 5 个 HUB 级联,最长扩展连接距离不得超过 30 m。第四,对大多数外设而言,USB 可以为它们提供足够的传输带宽。USB 允许两种数据传送速度。低速传送为 1.5 Mb/s,全速传送为 12 Mb/s(USB 1.0)。该速率与一个标准的串行端口相比,大约快 100 倍,与一个标准的并行端口相比,也快近 10 倍。因此,对需要使用高速通信接口(例如 ISDN、ADSL、E1 等)的用户,USB 也能提供足够的传输带宽。对更高带宽的需求,目前 USB 2.0 已可以提供 480 Mb/s 的传输速度。

2. USB 的优点

USB 的优点来自于其设计目标,这包括:易于使用、对用户隐藏技术实现细节、可以应用于不同的领域、具有足够的带宽以适应多媒体应用的要求、具有高可靠性、设备与系统相互独立。这些设计目标导致了 USB 具有以下优点:

1) 易于使用

易于使用是 USB 的主要设计目标,结果是 USB 接口受到设计人员和最终用户的广泛欢迎,其主要原因包括:

① 适合多种设备 USB 是一种通用的接口,可以适用于多种外设。这样就无需为每个外设准不同的接口和协议 一种接口就能满足多种外设。

② 自动配置 即插即用(PnP)。当用户连接 USB 外设到一个正在运行的系统时,Windows 能自动检测外设,加载合适的驱动程序。外设第一次连接到系统时,Windows 可能会提示用户插入驱动程序磁盘,但是除此以外,其他安装步骤都是自动的。不需要定位并运行安装程序,也无须重启系统。

③ 不需要用户设定 USB 不需要用户进行初始设置,例如端口地址和中断请求(IRQ)线等,这给使用带来了很大的方便。

④ 节省硬件资源 PC 上可供使用的 IRQ 线是一种宝贵的稀缺资源,无法给新的外设分配 IRQ 常常是使用 USB 的原因之一。如果外设都尽可能地使用 USB,就可使 IRQ 线空闲出来供那些必须使用 IRQ 的外设使用。对 USB 来说,它只需要若干个端口地址和一根 IRQ,而挂接到 USB 上的外设不需要其他任何资源。对比之下,每个非 USB 外设都要求有自己的端口地址,通常还要一根 IRQ 线,有时还要有一个扩展槽(例如 MODEM 卡)。

⑤ 易于连接 有了 USB,就不需要再打开计算机的机箱去为每个外设增加扩展卡。USB 的连接器和电缆都有确定的规格,即使没有经验的用户也不会接错。一个普通的 PC 机有 2~6 个 USB 端口,如果需要的话,还可以通过连接一个 USB 集线器来扩展端口的数量。集线器可以提供最多 7 个端口来连接更多的外设或集线器。一个 USB 可支持多达 127 个物理外设。

⑥ 简易电缆连接 USB 的设计非常易于连接。USB 电缆只有 4 根芯线:2 根数据线,一根电源和一根地线。电缆可长达 5 m。通过集线器,连接还可以扩展到 30 m。

⑦ 热插拔 可以在任何时候连接和断开外设,不管系统和外设是否开机都不会损坏 PC 或外设。当外设连接到 PC 上时,操作系统会自动检测到并准备使用。

⑧ 不需另备电源。USB 接口自带了电源线和地线,可以提供 +5 V 的电源供应。一个外设如果需要中等功率的电源供应(最多 500 mA),则它完全可以从总线得到电源供应而不需要使用外置电源。相比之下,大部分使用其他接口的外设将不得不选择要么在设备中自带电源,要么使用一个非常不方便的外置电源。

2) 速度较快

一个全速 USB 接口可以以 12 Mb/s 的速度进行通信。实际数据传输速率比这个数值要低一些,这是因为所有外设都共用总线,导致总线除传输数据外,还必须携带状态、控制和错误检测信号。当只有一个设备通信时,最大理论传输数据速率可达 9.6 Mb/s(1.2 MB/s)。如果这还不够快,USB 2.0 规范将允许以 480 Mb/s 传输数据。这使得 USB 对打印机和其他需要快速传递大容量数据的外设更具吸引力。USB 也支持 1.5 Mb/s 的低速传输。低速外设通常很便宜,而且它们的电缆可以更灵活(如鼠标),因为电缆不需要屏蔽。

3) 可靠性高

USB 的可靠性来自于硬件设计和数据传输协议两方面。USB 驱动器、接收器和电缆的硬件规范消除了大多数可能引起数据错误的噪声。此外,USB 协议采用了差错控制/缺陷发现机制,当检测到错误时能通知发送方重新发送前面的数据。检测、通知和重发都由硬件来完成,不需要任何软件的介入。

4) 低成本

虽然 USB 比以前的接口更复杂,但它的组件和电缆并不昂贵。带有 USB 接口的设备与带有相同功能的老式接口的设备所需的费用几乎是相同的,甚至更低。对成本非常低的外设来说,可以选择低速传输以降低对硬件的要求,使成本控制在合理的范围内。

5) 低功耗

当 USB 外设不被使用时省电电路和代码会自动关闭它的电源,但仍然能够在需要的时候做出反应。降低电源消耗除了可带来保护环境的好处之外,这个特征对于电源供应非常敏感的笔记本电脑尤其有用。

3. 主要技术指标

到目前为止,USB 已有两种版本,USB 1.1 和 USB 2.0。这两种版本的 USB 均采用一条 4 芯的电缆连接主机和 USB 设备。连接电缆除提供信号线外,还向 USB 设备提供了电源。USB 的主要技术指标见表 4-10。

表 4-10 USB 的主要技术指标

指标项	USB 1.1	USB 2.0
连接的设备数/台	127	127
传输速率 Mb/s	慢速: 1.5 全速: 12	慢速: 1.5 全速: 12 高速: 480
连接电缆长度/m	慢速: 3	慢速: 3
	全速: 5	全速: 5
电源供应	电压: 5 V 最大电流: 100 mA/500 mA	电压: 5 V 最大电流: 100 mA/500 mA

注: 每个单元负载为 100 mA,不同类型的设备可从 USB 获取 1~5 个单元负载的电流

4. USB 的总线拓扑结构和连接形式

USB 总线的拓扑结构是一种多层的星形结构,见图 4.24。每个星形的中心是一个集线器。一个集线器可以有 2~7 个端口,每一个端口都可以连接一个功能设备或另一个集线器。所有的连接都是点对点的。

USB 设备可以划分成两大类:集线器(Hub)和功能部件(Function)。只有集线器有能力提供附加的 USB 接入点。而功能部件为主机提供附加的功能。有些 USB 设备即是功能部件,也可以提供集线器功能,或称为复合设备。

图 4.25 所示为一台具有两个 USB 接口的 PC 可能的设备连接情况。如果只有两个 USB 外设,那么可以把它们分别插到 PC 上的两个 USB 端口中。如果最多有 6 个外设,那么可以

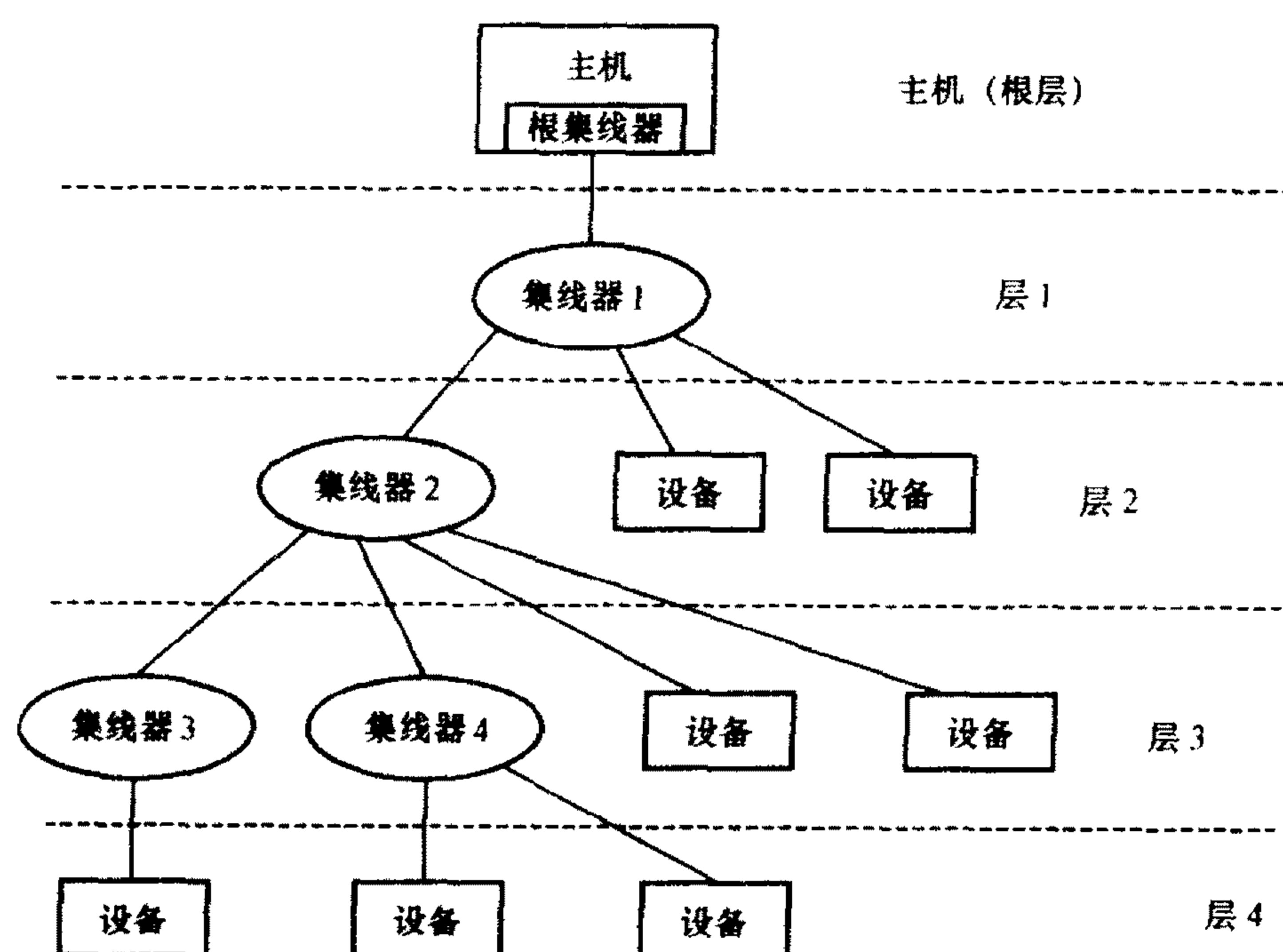


图 4.24 USB 总线逻辑拓扑结构

把一个复合设备插到 PC 的一个 USB 端口, PC 的另一个 USB 端口上连接一个带有 4 个扩充接口的 USB 集线器。然后把其他 4 个设备连接到集线器, 最后一个设备接到复合外设的扩充接口上。

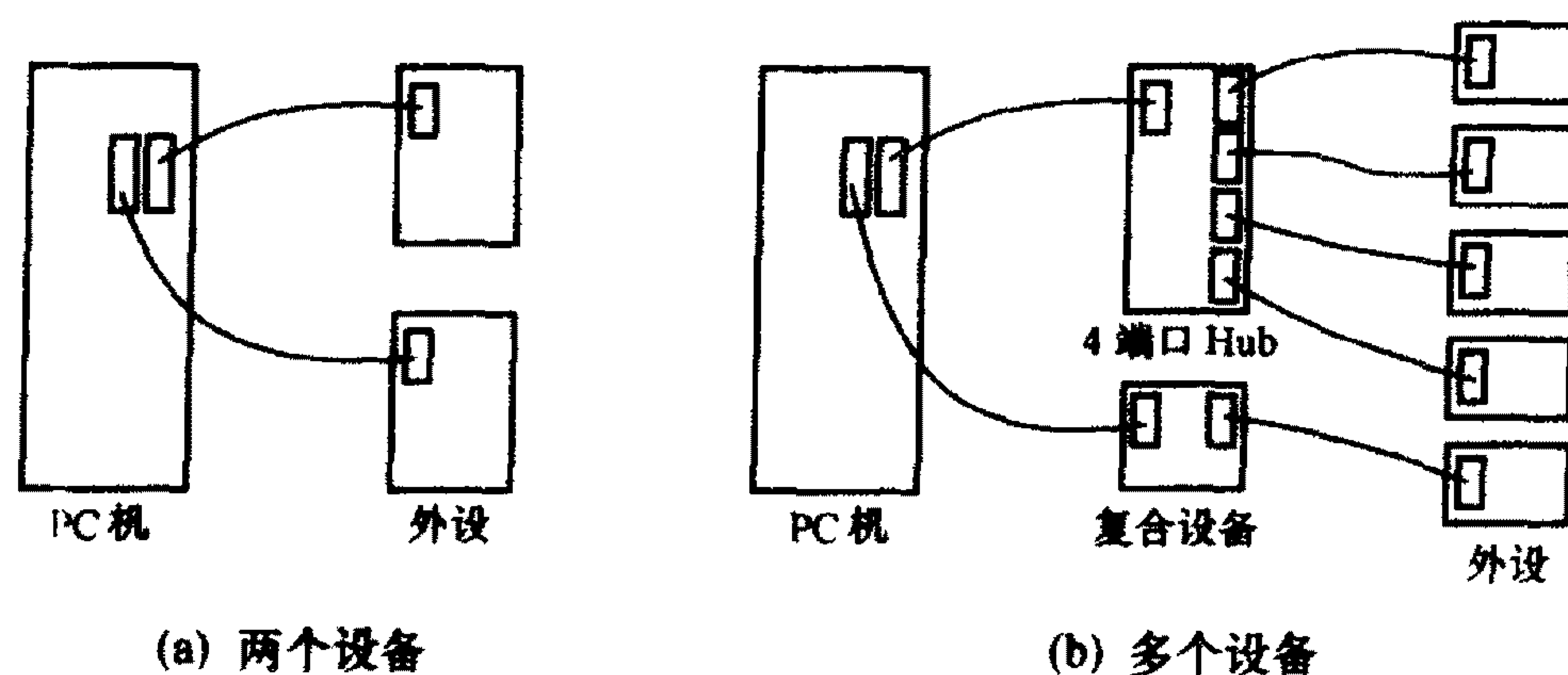


图 4.25 USB 总线的设备连接方法

图 4.26 所示是采用 USB 总线连接设备的实际例子。

1. USB 主机

在整个 USB 系统中只允许有一个主机。主机中的 USB 接口称为 USB 主控制器。而根

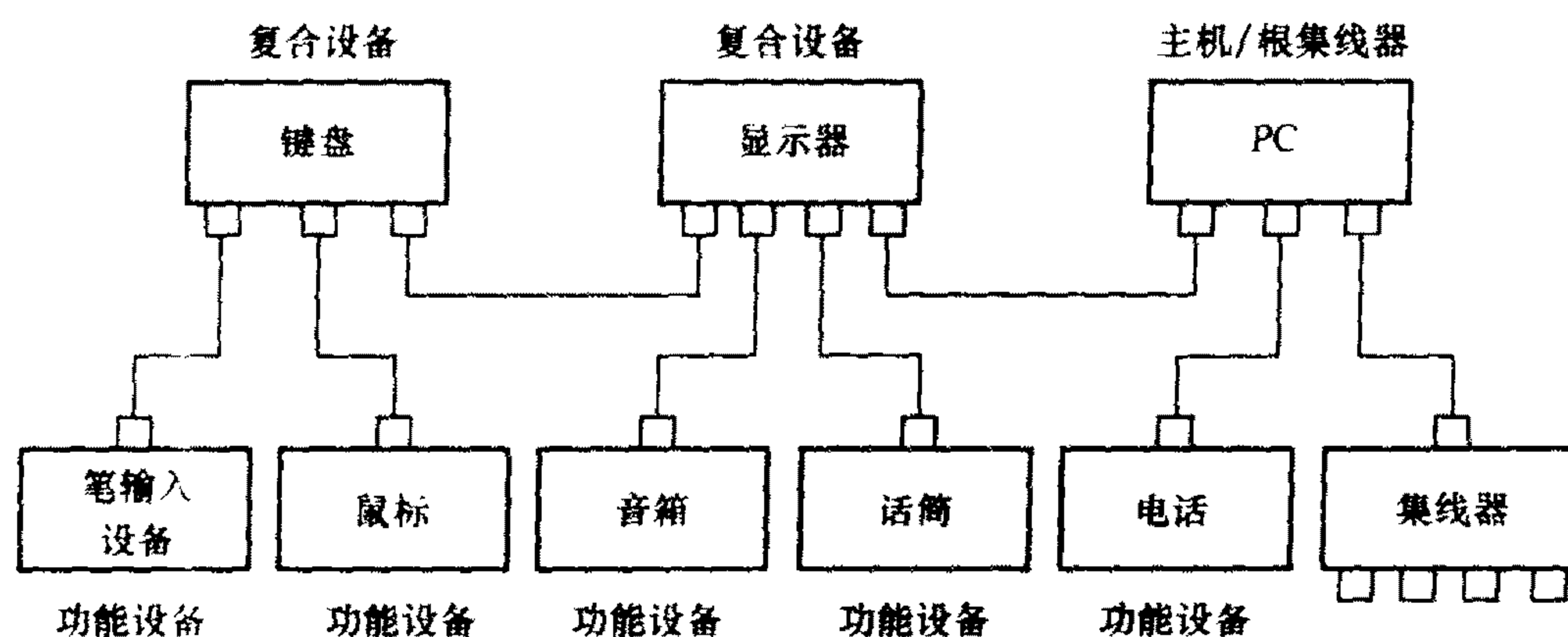


图 4.26 采用 USB 总线进行设备连接的实例

集线器是集成在主机系统中的。在 USB 规范中,USB 主机被定义为控制 USB 的软件和硬件的集合。如果不是很严格的话,可以认为 USB 主机就是 PC 机的硬件和相应的驱动程序。主机对 USB 的操作是:

1) 检测 USB 设备的插入和移出

在上电时,主机必须识别所有已经连接的 USB 设备。在识别过程中,主机为每个设备分配一个地址,并从设备获取其他配置信息。在上电后,无论何时当一个设备被连接或断开时,主机都能察觉到该事件的发生,并向设备表中加入任何新连接的设备或删除任何断开的设备。

2) 在主机与 USB 设备之间管理数据流

主机负责控制总线上的数据流。一般来说,USB 通信总是由主机发起,并由主机管理整个数据传输过程。

若多个外设希望同时传输数据,主机控制器按如下方式处理这个问题:它把数据通道分为 1 ms 的帧,然后给每个设备分配每一帧的一部分。

主机应该保证那些必须以固定速率进行的传输在每一帧中确实能得到它们所需要的时间量。在设备识别过程中,设备驱动将申请传输所需要的带宽。如果无法分配所需的带宽,主机就不允许与设备进行通信。不需要固定速率的传输使用帧的其他部分,并且可能需要等待。

3) 进行错误检查。

主机也有错误检查的责任。它往发送的数据中加入错误校验码。当设备收到数据时,按校验算法对数据执行计算,然后把结果与接收到的错误校验码进行比较。如果二者不同,则设备就不确认所收到的数据,于是主机便重新发送(USB 也支持不允许重新发送的传输类型,目的是保持一个稳定的传输速率)。主机对从设备接收到的数据也采用相同的方法进行

错误处理。

主机也可能收到其他错误指示符,指示设备现在不能发送/接收数据,这时主机就通知应用程序以采取合适的动作。

4) 提供电源

除了两个信号线,USB 还有一个 +5 V 的电源线和地线。大部分外设都可以从该线上得到所有所需的电源。在上电或连接时,主机给所有设备提供电源,当需要时还可以使这些设备工作在省电模式下。每个满负荷供电的设备需要高达 500 mA 的电流。在一些电池供电的 PC 的端口和集线器上只支持低功耗的设备,它们的工作电流被限制在 100 mA 以内。若设备自己有独立的电源供应,则只在刚开始与主机通信时使用总线电源。

2. USB 设备

USB 设备分为集线器和功能设备。图 4.27 给出了一个典型的集线器的示意图。集线器具有一个上行端口(Upstream Port)和若干个下行端口。上行端口用于连接主机或上级集线器,下行端口用于连接下级集线器或直接连接设备。通过集线器可实现 USB 总线的多级连接。在连接到 USB 总线的初期以及电源断开与重新接通时,除上行端口

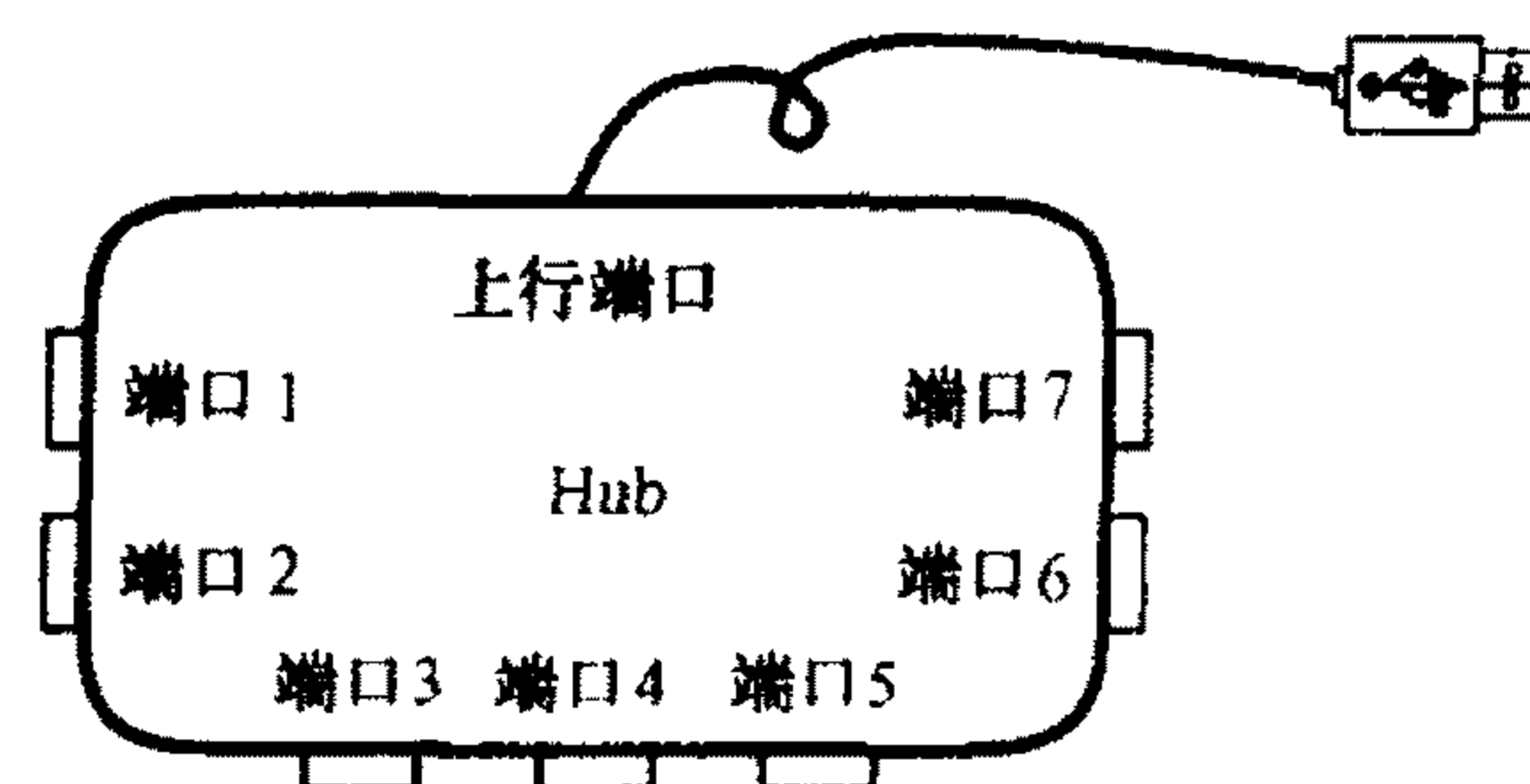


图 4.27 典型的 USB 集线器

外的所有其他端口是不能使用的(当然它们也可以被单独设置为可用或不可用)。另外,集线器可以发现下行端口上的设备插入或移出操作,并为下行设备分配电源。每一个下行端口都可以分别配置为全速或低速,集线器可以把低速端口与全速率信号分离开来。

功能设备是指一个可以从 USB 总线上接收或发送数据或控制信息的 USB 设备。一个功能设备由一个独立的外围设备实现,它通过一根电缆接到集线器上的某一端口。但是,一个物理组件也可以包含多个功能设备和一个嵌入式集线器,而且仅用一根 USB 电缆连接到上级集线器,这被称为复合设备。对主机而言,复合设备呈现为永远都连接着一个或多个 USB 设备的集线器。

每一个功能设备都包含了用来描述其能力和所需资源的配置信息。在使用一个功能设备之前,必须由主机来对其进行配置。这种配置操作包括分配 USB 带宽和为该功能设备选择特定的配置选项。功能设备的例子包括鼠标、键盘、打印机、MODEM、活动硬盘、数字相机等。

与主机不同的是,设备不能主动发起一次 USB 通信,只能等待主机并响应主机发起的通信(一个例外是远程唤醒特征,这个特征使一个设备能向主机申请通信)。

设备对 USB 的责任是:

1) 检测与自己的通信

每个外设始终监测着总线上的通信,若通信的设备地址与设备本身的地址不同,则设备

忽略这次通信。如果地址相同,设备就把通信的数据保存在它的接收缓冲器中,并产生一个中断来发出数据已经到达的信号。在几乎所有的芯片中,这个步骤都是自动完成的,它被内置于硬件中。

2) 对标准请求的响应

在上电和连接到带电系统时,设备必须在识别过程中对主机发出的请求做出响应。当收到请求时,设备把要发送的响应信息放置在它的传输缓冲器中。在某些情况下,如设定一个地址和配置,设备除了发出响应信息外还要进行其他动作。然而设备不必执行每一个请求,它只需要以一种可以理解的方式对请求做出响应。例如,当主机请求使用一个设备不支持的配置时,设备用一个指示符来响应,通知主机该请求不被支持。

3) 错误检查

像主机一样,设备在要发送的数据后面加入错误校验码。在接收到数据时,设备进行错误校验计算,如果检测到错误就请求重新传输。这些功能内置在硬件中,不需要软件实现。

4) 管理电源

如果设备不从总线获得电源供应,它就必须自己供电。当没有总线活动时,设备必须进入它的低功耗挂起状态,但继续监视总线,当总线活动恢复时退出挂起状态。

当主机进入低功耗状态时(如 Windows 98 的待机状态),所有与总线的通信都停止了。与之相对应,连接到总线的设备检测到总线活动停止了 3 ms 时,它们也必须进入挂起状态并且限制从总线中获取电流。主机也可能请求挂起与一个特定设备的通信。当总线活动恢复后,设备必须退出挂起状态。

不支持远程唤醒特征的设备在挂起状态下从总线取出的电流不会超过 500 mA。有远程唤醒特征的设备并且该特征被主机使能后,这个极限是 2.5 mA。

5. USB 的连接

USB 规定了两种连接器,分别称为 A 系列和 B 系列,如图 4.28 所示。通常计算机主机板上的连接器都是 A 系列的,B 系列的连接器常用于设备端上。对键盘、鼠标和扩充集线器等 USB 设备,其引出线上的插头多使用 A 系列连接器与主机实现连接。

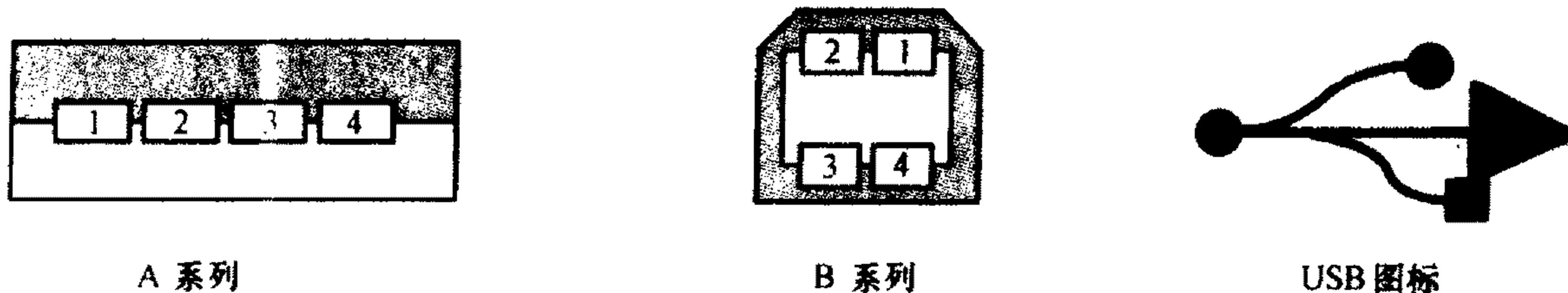


图 4.28 两种常见的 USB 连接器的前视图

USB 连接器有 4 根引脚,各引脚信号的定义和用途见表 4 - 11。

表 4 - 11 USB 的引脚配置

引脚号	信号名称
1	+ 5 V
2	信号 (负)
3	信号 (正)
4	地线

6. USB 数据信号

USB 的数据信号为双相信号,它使用一个差模输出驱动器来产生。图 4.29 给出了一个全速率驱动器的实例。图中的 SN75240 为噪声抑制电路。

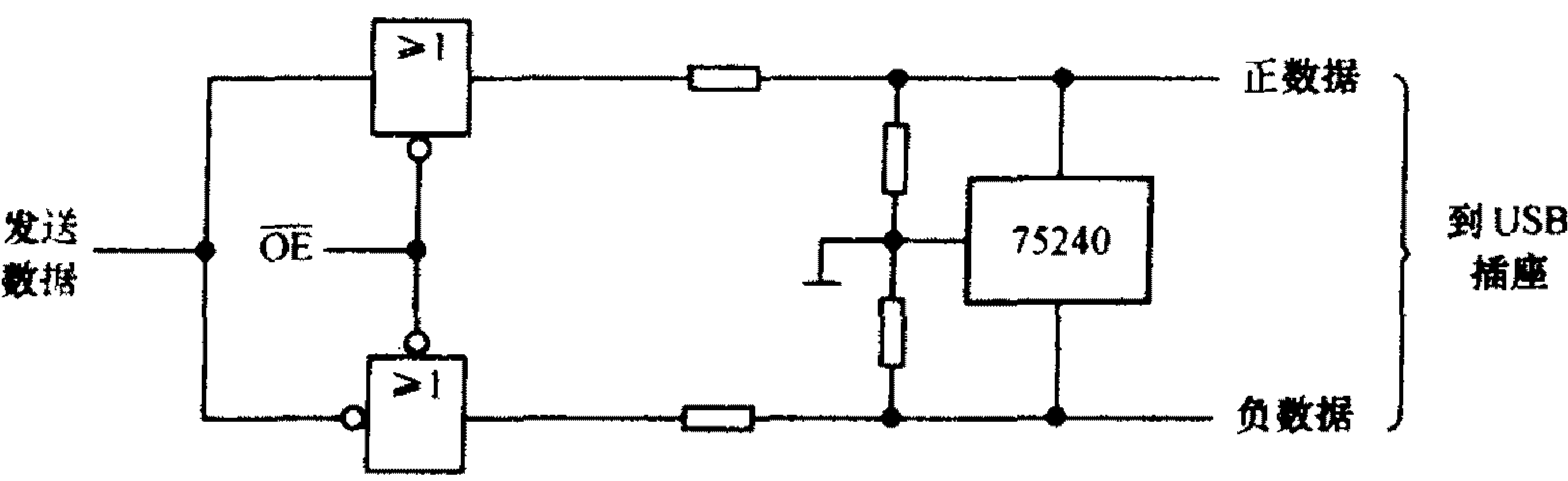


图 4.29 全速率 CMOS 驱动器电路实例

发送的数据经 NRZI(Non - Return to Zero, Inverted, 反向不归零制)编码后再放到总线上传输。在 NRZI 编码方式中,电平不变化表示逻辑“1”,而电平发生变化表示逻辑“0”。图 4.30 给出了一个数据位流和用 NRZI 编码后产生的 USB 信号。在 NRZI 编码过程中,一连串的“0”会使得 NRZI 数据每个位周期都会出现跳变;而一连串的“1”则使得数据中长时间不会出现变化,这会造成信号中产生直流成分,并可能使收发双方失去同步。

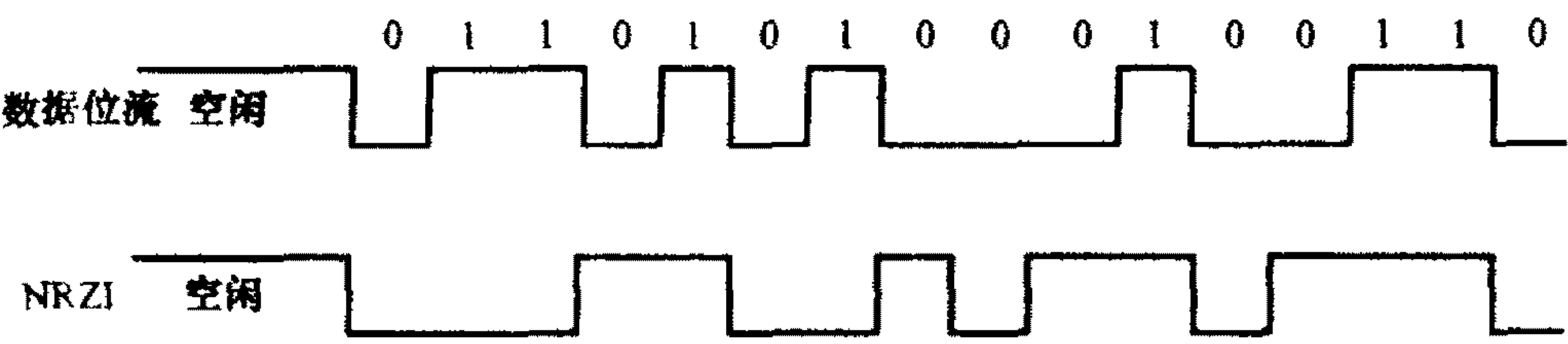


图 4.30 NRZI 数据编码

为了保证信号有足够的变化,在 USB 上发送数据时,传输设备要进行位填充(bit stuffing)。位填充是在 NRZI 编码之前进行的。位填充的方法是在数据位流内每 6 个连续的“1”后都插入一个“0”,从而在 NRZI 数据流中强制加入一个变化。这样在逻辑上至少每 7 个位周期,接收器就会收到一个数据变化,以保证数据和时钟相互锁定。位填充总是被强制执行,从而确保了对于长的逻辑“1”串,接收器也会保持同步。对于接收方,接收器必须对 NRZI 数据进行解码,识别填充位并丢弃它们,以恢复原始的位流。

7. USB 通信流

图 4.31 给出了 USB 通信模型的基本通信流和相互关系。

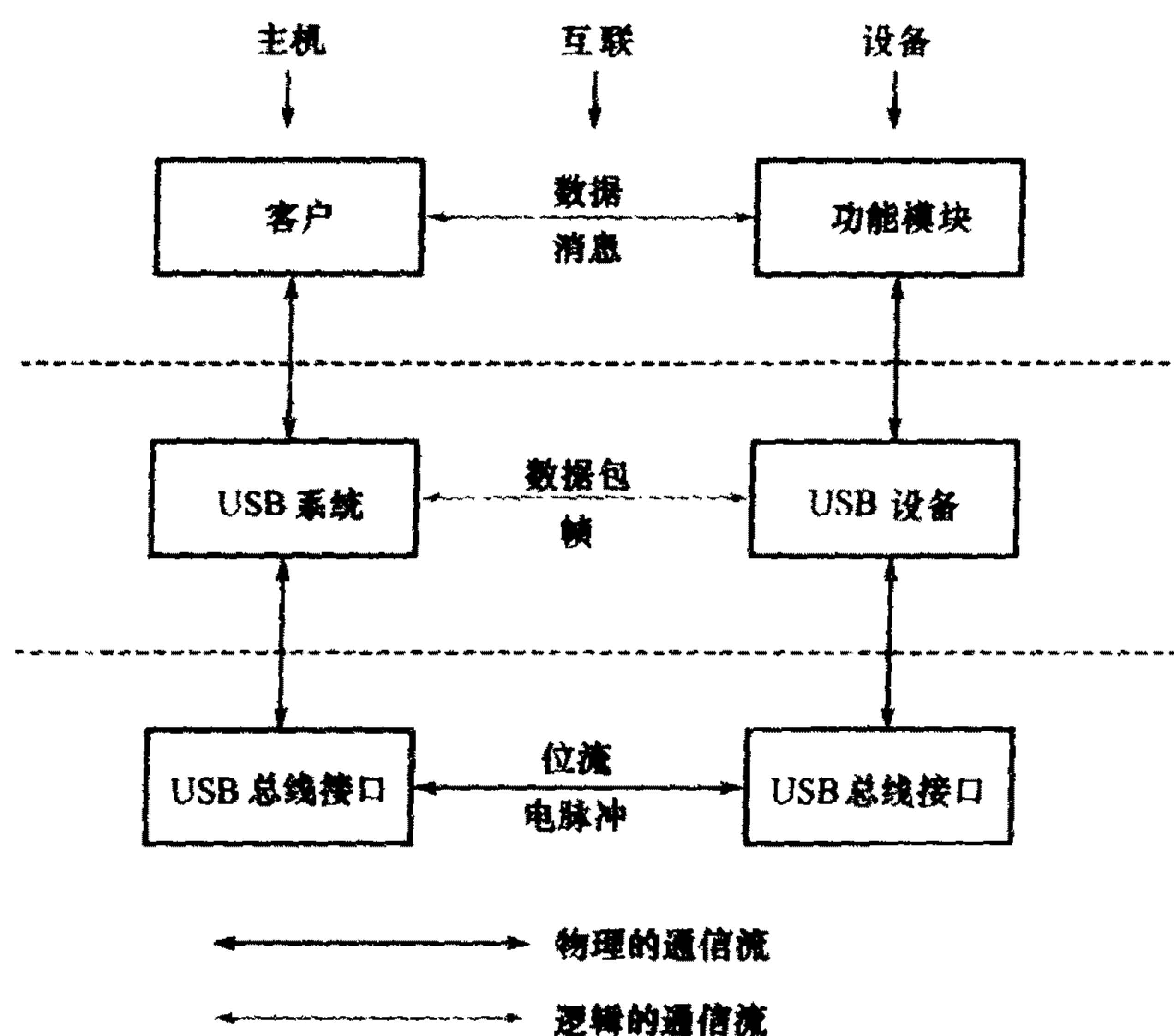


图 4.31 USB 通信模型：逻辑通信与物理通信

主机和设备都分成图 4.31 中所描述的清晰的层次。每一层的实体称为对等实体。对等实体间进行的通信是逻辑上的通信,逻辑通信是用协议来约束的。主机和设备上的实际通信用垂直箭头指出,每一个垂直箭头都表示相应两层之间的接口。在主机和设备之间进行的任何逻辑通信,最终都必须出现在物理的 USB 电缆上。

主机上的软件通过一系列的通信流来与一个逻辑设备进行通信。为了更详细地描述 USB 通信流,图 4.32 给出了一个 USB 主机与设备的详细视图。从图中可以看出,客户软件使用的是管道束(与一个端点集对应)来与功能模块通信,USB 系统软件使用缺省管道(与端点 0 对应)来管理配置 USB 设备。而真正的物理通信则是出现在 USB 电缆上。

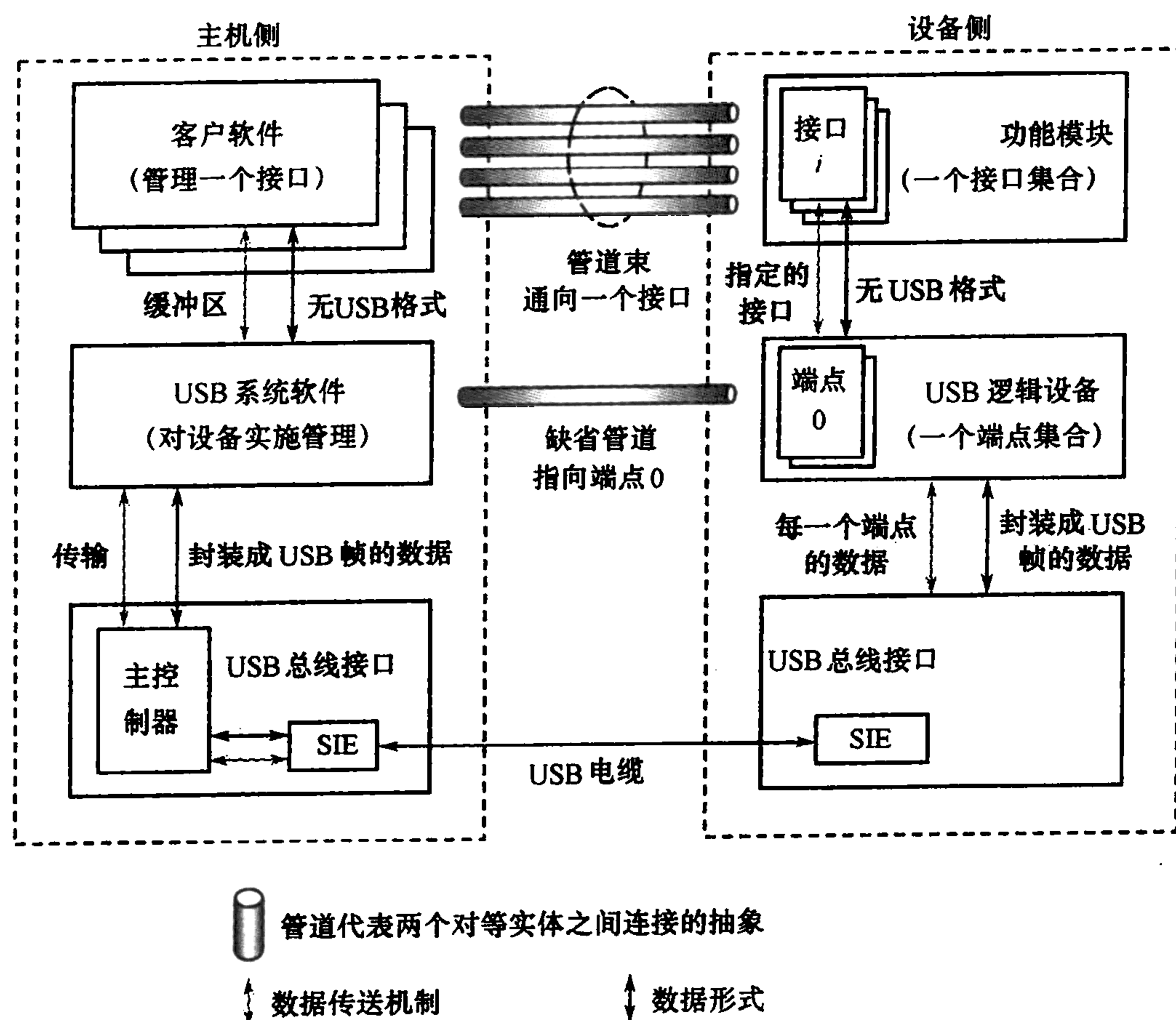


图 4.32 USB 主机/设备的详细图示

下面对图中涉及的端点和管道的概念进行解释。

1) 设备端点(End Point)

端点是一个 USB 设备惟一可以确认的部分,它是主机和设备之间通信流的终点。USB 为主机上的客户软件与 USB 功能模块之间的通信提供了服务。不同的功能模块会对通信流有不同的要求(例如,USB 活动硬盘的设备配置功能与文件读/写功能具有完全不同的通信流)。为了提供更好的整体总线利用率,USB 把不同的 USB 功能模块的不同通信流进行了分离,每一个通信流都终止于逻辑设备上的某一个端点。

每一个 USB 逻辑设备都包含了一个相对独立地进行操作的端点集合。软件只能通过一个或多个端点与一个 USB 设备通信。在这个意义上,任何一个通信流都可以用一个设备端点来识别。在设备接入时,系统会为每一个逻辑设备分配一个惟一的地址。而逻辑设备上的任意一个端点都有一个在设计时就已确定的惟一的标识和端点号。于是利用设备的地址和端点号就可以惟一地指定任何一个端点。

端点可以决定端点和客户软件之间通信所需要的传输服务类型。一个端点具有以下

些属性:

- ① 端点号;
- ② 总线频率/延时要求;
- ③ 带宽要求;
- ④ 差错控制要求;
- ⑤ 端点可以接收或传递的最大分组;
- ⑥ 端点的传送类型;
- ⑦ 端点和主机之间的数据传送方向。

端点 0 是 USB 设备的缺省端点,所有 USB 设备都必须拥有端点 0,该端点用于对一个逻辑设备进行配置(初始化)。端点 0 提供了对设备配置信息的访问权,通过它还允许访问 USB 状态和控制操作。端点 0 总是在设备接入和上电时就立即进行配置。

除端点 0 外,功能设备还具有其他的端点。低速功能设备有两个端点可供选择。而对于全速率设备来说,它的附加的端点数仅受到协议的限制,最多可有 16 个输入端点和 16 个输出端点。

在对端点进行配置之前,端点处于一种不确定的状态。所以一个端点只在对其进行配置之后,主机才能访问它。包括端点 0 在内的所有端点,都作为设备配置过程中的普通对象来对其进行配置。

2) 管道

一个 USB 管道是设备上的一个端点和主机上的软件的联合体。管道表示经过一个存储器缓冲区和一个设备上的端点,可以在主机上的软件之间传送数据的能力。

USB 并不对管道中传递的数据内容进行翻译,即使是消息管道要求根据 USB 的规定对数据进行打包,USB 也不会翻译这些数据的内容。

对一个 USB 设备进行配置之后,就会形成管道。由于一个 USB 设备上电后总要对端点 0 进行配置,所以端点 0 总是拥有一个管道。该管道称为缺省管道。系统软件利用该管道来识别设备和确定配置要求,并对设备进行配置。当设备配置完毕后,该设备的专用软件也可以使用缺省管道。同时 USB 系统软件也保留对缺省管道的“所有权”,并由它来协调其他客户软件对该管道的使用。

管道分为流管道和消息管道两种:

① 流管道是指通过管道的数据不具有确定的 USB 定义的结构。数据流从一端进入总线,并以同样的顺序从另一端流出。而且在通信流中流管道通常是单向的。

通过一个流管道进行数据传输时,多个请求决不会同时送达一个端点,流管道机制对多个同时的请求总是进行串行化处理。所以 USB 系统软件不必在可能使用同一个流管道的多个客户之间提供同步机制。进入管道中的数据以先进先出方式流向对方。

通往某个设备的一个流管道,在一个方向上必须与一个惟一的设备端点号相对应(即与协议层所指定的一个 IN 或 OUT 令牌所对应)。用于反方向的设备端点号也可由通向该设备的另外的某个流管道使用。流管道支持批量、同步和中断三种数据传输类型。

② 消息管道是指通过管道的数据具有 USB 定义的某种结构。消息管道用与流管道完全不同的方式来与端点进行沟通。首先,主机向该 USB 设备发出一个请求,该请求后面紧跟着是某一方向的数据传输,然后在某一时刻端点会返回一个状态作为响应。为了适应这种“请求”/“数据”/“状态”对话模式,消息管道要求通信流具有一定的结构,这样,命令就可以被可靠地识别和传输。虽然通信流多数情况下是单方向的,但消息管道却允许双向的数据流存在。与端点 0 对应的管道,即缺省管道,总是一个消息管道。消息管道支持传输控制类型的通信流。

图 4.33 说明了通信流是如何在主机一侧的缓冲区与设备一侧的端口之间的管道中流动的。

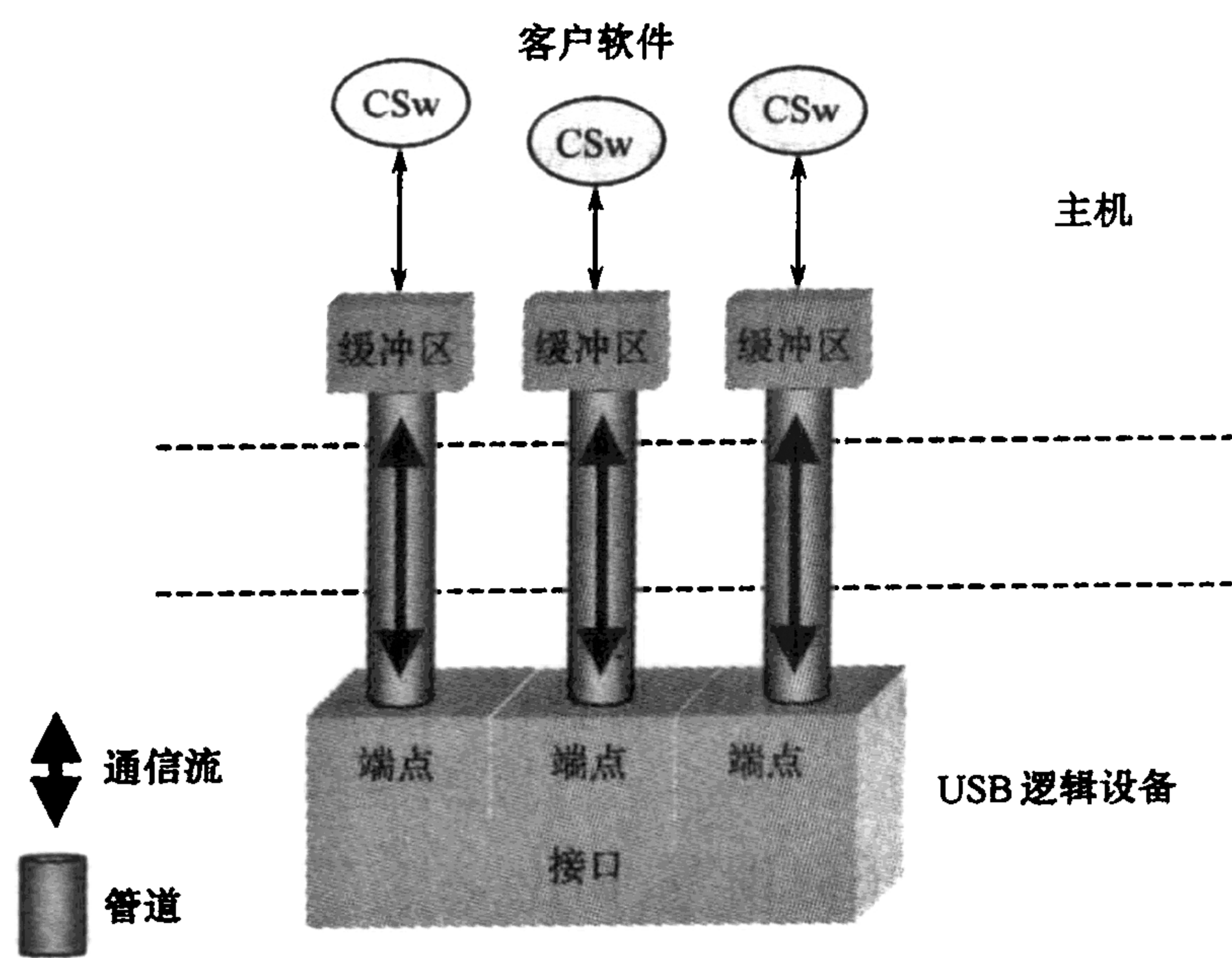


图 4.33 USB 通信流

8. USB 的传输类型

USB 定义了以下四种传输类型:

① 控制传输 主要用于命令/状态操作。它是由主机软件发起的请求/响应通信过程,具有突发性,非周期的特点。

② 同步传输 主要用于主机和设备与时间有关的信息传输。具有周期性、连续性的特点。这种传输类型保留了数据中时间压缩的概念。但是它并不意味着这一类数据传送都是

实时的。

③ 中断传输 主要用于向主机通知设备的服务请求。它是由设备发起的通信,具有数据量小、非周期、低频率、延时固定等特点。

④ 批量传输 主要用于那些可以利用任何可用的带宽进行传送,或可以延迟到有可以利用的带宽时再进行传送的数据。它具有非周期和突发性强的特点。

9. USB 主机的软件和硬件

每个 USB 中只有一个主机。USB 主机是 USB 中惟一一个用于协调工作的实体。对 USB 的访问都是由主机控制,只有当主机允许时,一个 USB 设备才能获得对总线的访问权。主机以一种分层结构来组成,的主要分层是:

- ① USB 总线接口;
- ② USB 系统;
- ③ 客户。

USB 总线接口控制了电气和协议层的交互。从互连的角度看,它呈现为一个串行接口引擎 (Serial Interface Engine, SIE)。但从主机的角度来看,它以主控制器的形式实现。

USB 系统利用主控制器提供的服务来控制主机和 USB 设备之间所进行的数据传输。USB 系统与主控制器之间的接口依赖于主控制器的硬件定义。USB 系统有 3 个基本的组件:主控制器驱动程序 (HCD), USB 驱动程序 (USB D) 和主机软件。它们之间的关系如图 4.34 所示。

① HCD 用于隐藏不同控制应用的细节,向上层提供统一的服务,从而使与设备进行交互的客户不必了解该设备是同哪一个主控制器相连。HCD 同 USB D 之间所存在的接口称之为主控制器驱动程序接口 (HCDI),此接口不对用户开放,而是由操作系统来使用。

② USB D 为客户使用 USB 设备提供了方便。USB D 以 I/O 请求分组的方式来提供数据传输机制,I/O 请求分组内包括了一个通过某个管道来传送数据的请求。除了提供数据传输机制之外,USB D 还负责向其客户提供一个 USB 设备的抽象,可以控制它来实现配置和状态管理。该抽象最重要的部分就是缺省管道,利用缺省管道可以对所有 USB 设备进行标准

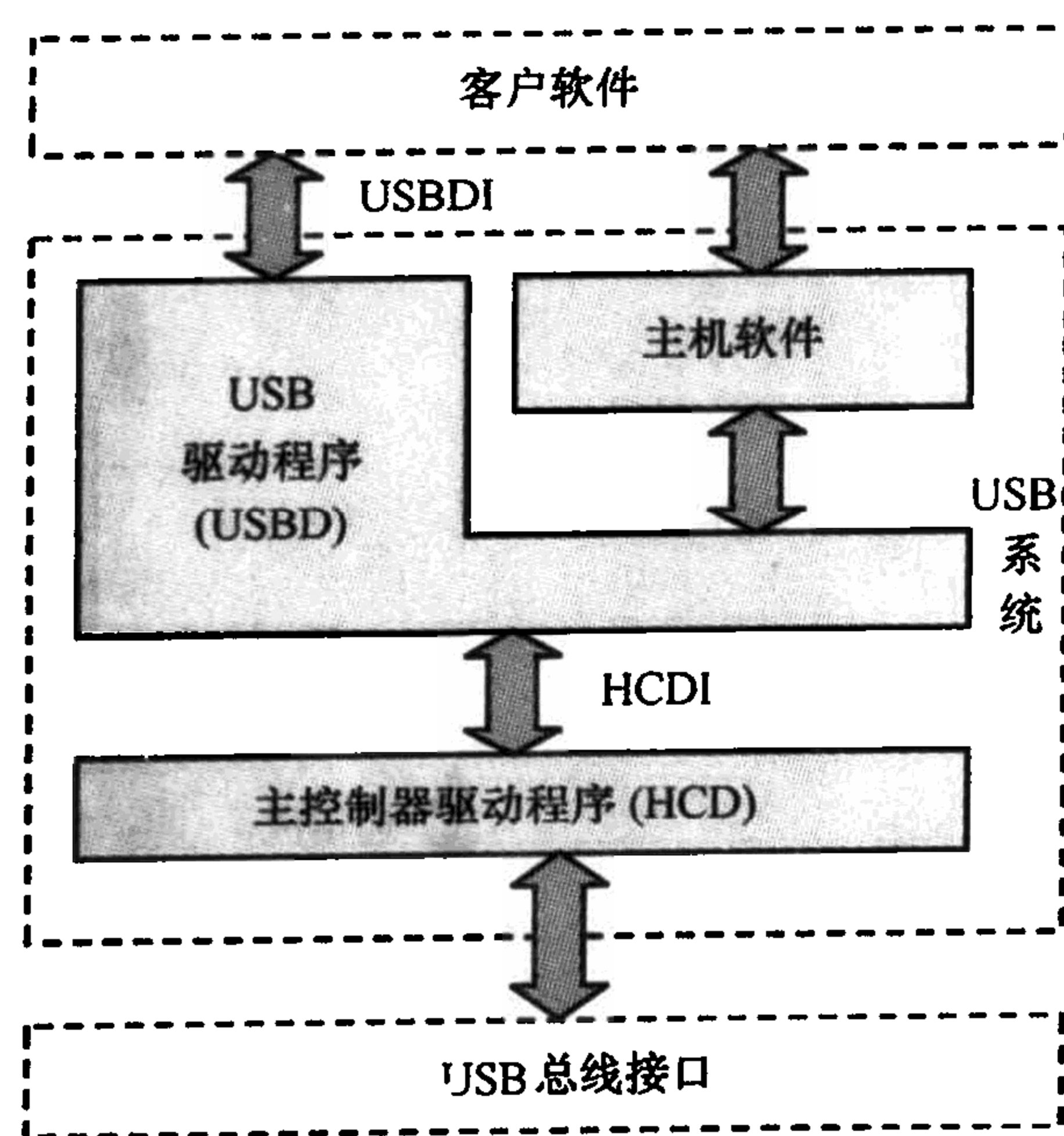


图 4.34 USB 系统层的组件

USB 配置。这一缺省管道代表了 USB 设备抽象之间所存在的一个逻辑通信。USB 与客户之间的接口称为 USB 驱动程序接口 (USB DI), 它是客户访问 USB 设备的主要接口。

③ 主机软件很重要的一个用途就是提供设备配置和加载机制。客户可使用主机软件提供的接口进行设备配置和加载, 以取代直接访问 USB DI 的机制。

④ 客户(注意, 这里所指的客户是一种抽象的概念, 它泛指能够从下层获得服务的软件实体)描述了所有负责与其相关外设打交道的软件实体。当一个设备接入系统时, 客户可以通过下面各层提供的服务来与设备硬件进行通信, 但不允许直接访问设备硬件(尽管在逻辑上客户是与设备的功能模块进行通信的)。

10. 总线枚举

总线枚举是指对总线上接入的 USB 设备进行识别和寻址操作。由于 USB 支持热插拔和即插即用, 所以当有一个 USB 设备接入 USB 或从 USB 上拆除时, 主机必须使用总线枚举的过程来识别和管理必要的设备状态变化。并动态地对它进行配置。当一个 USB 设备接入后, 下列事件将会发生:

① USB 设备所接入的集线器通过一个其状态变化管道上的响应向主机报告该事件。这时, USB 设备处于连接状态, 而连接它的端口则被禁用。

② 主机通过询问集线器来确定变化的真实性质。

③ 主机现在已经知道了新的设备所接入的端口, 于是向该端口发送一个端口激活和复位信号。

④ 集线器把发往该端口的复位信号保持 10 ms。当复位信号释放后, 被激活的端口和集线器将向 USB 设备提供 100 mA 电流。现在 USB 设备就处于加电状态。它的所有寄存器和状态都被重新设置, 而且它可以对缺省地址做出响应。

⑤ 在为该 USB 设备分配地址之前, 利用缺省地址仍然可以访问其缺省管道。主机通过读取该设备的描述符, 来确定这一 USB 设备的缺省管道实际可以使用的最大数据负载尺寸。

⑥ 主机为 USB 设备分配一个惟一的 USB 地址, 然后用这个地址和端点 0 来建立该 USB 设备的控制管道。

⑦ 主机读取设备的每一项配置信息。这个过程可能需要传输若干个帧的数据。

⑧ 根据配置信息, 主机就知道如何来使用这一 USB 设备, 于是主机向设备分配一个配置值, 这时设备就处于配置完成状态, 并且在这一配置中的所有端点都具有其描述的特征。现在 USB 设备就可以获得其配置描述符中所描述的额定电流量。从设备的观点来看, 它已经为使用做好了准备。

当 USB 设备被拆除时, 集线器也会通知主机。拆除一个设备会使该设备所接入的端口

被禁用。一旦收到了拆除指示,主机将立即更新它的本地拓扑结构信息。

11. USB 传输与数据包格式

USB 传输数据的格式与计算机网络传输数据的格式非常相似,即所有的数据都必须封装成帧(或称为数据包)才能递交给总线接口送到总线传输。任何数据包发送前,都要先发送一个同步字节(80H),然后紧接着发送数据包。数据包的第一个字节是数据包识别字节(PID)。PID 的定义见表 4-12。

表 4-12 PID 代码

PID 字节	名称	类型	描述
E1H	OUT	标记	主机到设备事务的端点地址
69H	IN	标记	设备到主机事务的端点地址
A5H	SOF	标记	帧起始标记和帧编号
2DH	Setup	标记	主机到设备的 Setup 事务的端点地址
D2H	ACK	信号交换	接收器接收到无错误的数据包
5AH	NAK	信号交换	接收器不能接收/发送数据或没有数据要发送
1EH	Stall	信号交换	一个控制请求不支持或端点被禁止
C3H	Data0	数据	有偶同步位的数据包
4BH	Data1	数据	有奇同步位的数据包
3CH	PRE	特殊	主机发送的前同步信号,允许到低速设备的下行通信

图 4.35 列出了 USB 传输中出现的数据、标记、信号交换以及帧起始数据包的格式。其中,地址字段包含了 USB 设备的 7 位地址(所以一个 USB 最多可寻址 127 个设备),端点号字段用于指示通信流的端点。CRC 字段是数据校验字段,USB 使用了两种 CRC(Cyclic Redundancy Check,循环冗余校验):CRC₅ 和 CRC₁₆。CRC₅ 的生成多项式为 $X^5 + X^2 + 1$ (对应的二进制数为 100101),CRC₁₆ 的生成多项式为 $X^{16} + X^{15} + X^2 + 1$ (对应的二进制数为 11000000000000101)。

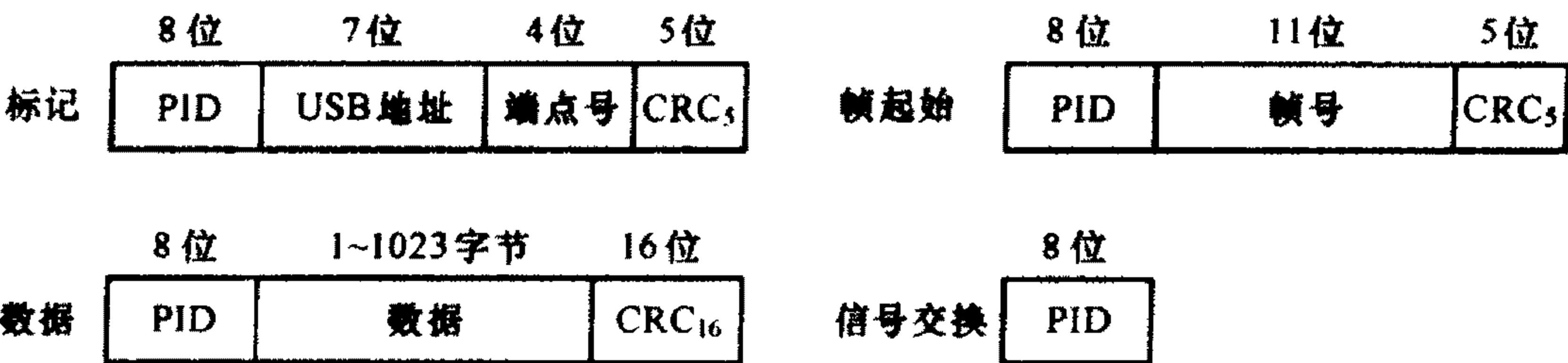


图 4.35 USB 数据包的格式

USB 使用 ACK 和 NAK 来协调数据包在主机系统和 USB 设备之间的传输。USB 设备一旦收到从主机发来的数据包,就应发回一个 ACK(确认)或 NAK(否认)给主机。如果数据被正确地接收,则发送 ACK;如果接收不正确,则发送 NAK。如果主机接收到 NAK,则它重新发送该数据包,直到接收器正确地接收到此数据包为止。这种数据传输的方法常被称为停等式数据流控制(Stop and Wait Flow Control)。这种方法的关键是,主机在传输下一个数据包之前,必须等待接收方返回对上一个数据包的回应信息。

12. USB 的现状与发展

由于 USB 的广泛应用,PC97/98 标准已经纳入了 USB 规范,新的芯片组都支持 USB^①,许多软硬件厂商也已正式支持 USB,并且推出了许多采用 USB 的产品。例如,支持 USB 的总线控制器已有 i930 微控制器系列,C5M 系列以及 CYC63X00 系列等。在系统软件方面,Microsoft 公司在 Windows 98 和 Windows NT/2000/XP 操作系统中全都内置了支持 USB 标准的模块,并且为 USB 开发了相应的驱动程序和支持软件;计算机厂商生产的新主板几乎都带有 2~6 个 USB 端口;不少外部设备厂商纷纷推出了带有 USB 端口的键盘、鼠标、活动硬盘、扫描仪、MODEM 和游戏操纵杆等。值得注意的是,USB 2.0 标准已在 2000 年 4 月正式公布,它的最高传输速率达到了 480 Mb/s,基本上可以满足目前大多数外设的要求。采用 USB 2.0 接口的数码相机、外置硬盘等产品也已推向市场。根据国际数据集团(IDG)的估计,在不久的将来,USB 将会取代现有的串口和并口,成为微型计算机外设接口的最重要的标准之一。

4.5.2 IEEE 1394 总线

IEEE 1394 是一种串行接口标准,这种接口标准允许把微型计算机、外部设备以及各种家电非常简单地连接在一起。从 IEEE 1394 可以连接多种不同外设的功能特点来看,也可以称为总线,即一种连接外部设备的机外总线。IEEE 1394 的原型是运行在 Apple Mac 微型计算机上的 Fire Wire(火线),最终被 IEEE 采用并且重新进行了规范化。它定义了数据的传输协定及连接系统,可用较低的成本达到较高的性能,以增强微型计算机与外设如硬盘、打印机、扫描仪,与消费性电子产品如数码相机、DVD 播放机、视频电话等的连接能力。

由于要求相应的外部设备也具有 IEEE 1394 接口功能才能连接到 IEEE 1394 总线上,所以直到 1995 年第 3 季度 Sony 推出的数码摄像机加上了 IEEE 1394 接口后,IEEE 1394 才真正引起广泛的注意。采用 IEEE 1394 接口的数码摄像机,可以毫无延迟地编辑处理影像、声音数据,其性能得到增强。数码相机、DVD 播放机和一般消费性家电产品,如 VCR、HDTV、音响等也都可以利用 IEEE 1394 接口来互相连接。微型计算机的外部设备,例如硬盘、光驱、

① Intel 公司的 845E/G 芯片组中的 ICH4 芯片已支持 USB 2.0 规范。

打印机、扫描仪等,也可利用 IEEE 1394 来传输数据。IEEE 1394 总线改变了当前微型计算机拥有众多附加插卡、使用各种杂乱的设备信号线进行设备连接的现状,它把各种外设和各种家用电器通过一个单一总线连接起来,而且达到了极高的数据传输速率。

1. IEEE 1394 的主要性能特点

IEEE 1394 具有以下几个特点:

1) 采用“级联”方式连接各个外部设备

IEEE 1394 在一个端口上最多可以连接 63 个设备,设备间采用树形或菊花链结构。设备间允许电缆的最大长度是 4.5 m,采用树形结构时可达 16 层,从主机到最末端外设总长可达 72 米。

2) 能够向被连接的设备提供电源

IEEE 1394 的连接电缆(Cable)中共有 6 条芯线。其中两条线为电源线,可向被连接的设备提供电源;其他 4 条线被包装成两对双绞线,用来传输信号。电源的电压范围是 8 ~ 40 V 直流电压,最大电流 1.5 A。像数码相机之类的一些低功耗设备可以直接从总线电缆内部取得电源供应,而不必为每一台设备配置独立的供电系统。由于 IEEE 1394 能够向设备提供电源,即使设备断电或者出现故障也不影响整个网络的运转。

3) 采用基于内存的地址编码,具有高速传输能力

总线采用 64 位的地址宽度(16 位网络 ID,6 位节点 ID,48 位内存地址),将资源看作寄存器和内存单元,可以按照 CPU 内存的传输速率进行读/写操作,因此具有高速的传输能力。IEEE 1394 总线的数据传输率最高可达 400 Mb/s,因此可以适用于各种高速设备。

4) 采用点对点结构(Peer-to-Peer)

任何两个支持 IEEE 1394 的设备可以直接连接,不需要通过微型计算机控制,例如在微型计算机关闭的情况下,仍可以将 DVD 播放机与数字电视机连接,直接播放光盘节目。

5) 安装方便且容易使用

允许在带电的状态下实现即插即用,不必关机即可随时动态配置外部设备,增加或拆除外设后 IEEE 1394 会自动调整拓扑结构,重新设置整个外设网络状态。

2. IEEE 1394 的工作模式

IEEE 1394 标准定义了两种总线数据传输模式,即: Backplane 模式和 Cable 模式。其中 Backplane 模式支持 12.5、25、50 Mb/s 的传输速率;Cable 模式支持 100、200、400 Mb/s 的速率。目前正在开发 3.2 Gb/s 的版本。在 400 Mb/s 时,只要利用 50% 的带宽就可以支持不经压缩的高质量数字化视频信息流。

IEEE 1394 可同时提供同步(Synchronous)和异步(Asynchronous)数据传输方式。同步传输应用于实时性的任务,而异步传输则是将数据传送到特定的地址(Explicit Address)。这一

标准的协议称为等时同步(Isosynchronous)。使用这一协议的设备可以从 1394 连接中获得必要的带宽。其余的带宽可以用于异步数据传输,异步数据传输过程并不保留同步传输所需的带宽。这种处理方式使得两种传输方式各得其所,可以在同一传输介质上可靠地传输音频、视频和计算机数据,并且对计算机内部总线没有影响。

3. IEEE 1394 与 USB 的异同

① IEEE 1394 和 USB 总线都属于串行外设总线,它们都可以在一根总线上支持多个设备的连接;

② 都可以提供即插即用及热插拔的功能;

③ 采用“级联”方式,可以通过 HUB 连接多台设备,避免了微型计算机背板仅能提供少量接口而对设备连接数量的限制;

④ 目前 IEEE 1394 规范的传输速度为 400 Mb/s,因此适合于连接高速设备如 DVD 播放机、数码相机、硬盘等。USB 分为两个版本,USB 1.1 仅支持 12 Mb/s 的传输速度,适合于连接低速的键盘、鼠标、软驱等设备;USB 2.0 支持 480 Mb/s 的传输速度,其适用范围与 IEEE 1394 相同,但目前使用 USB 2.0 的外设还很少。

⑤ IEEE 1394 的拓扑结构中,不需要集线器(Hub)就可连接 63 台设备,并且可以由网桥(Bridge)再将这些独立的子网(Subtree)连接起来。IEEE 1394 并不强制要用微型计算机来控制这些设备,也就是说这些设备可以独立工作。而在 USB 的拓扑结构中,必须通过 Hub 来实现多重连接,每个 Hub 有 7 个连接头,整个 USB 网络中最多可连接 127 台机器,而且一定要有微型计算机的存在,作为总的控制。

⑥ IEEE 1394 的拓扑结构在其外部设备增减时,会自动重设网络,其中包括网络短暂的等待状态;而 USB 以 Hub 来判明其连接设备的增减,因此可以减少 USB 网络动态重设的状况。

总的来说,USB 和 IEEE 1394 在功能和设计思想上有许多相似的地方,但是它们的传输速率不同,因而适用范围也不同。

习 题 四

4.1 什么是总线?它的基本功能是什么?

4.2 CPU 总线包括哪 3 个组成部分?

4.3 总线数据传送有哪几种定时方式?各自的优缺点是什么?

4.4 在微型计算机中是如何解决多个部件争用总线问题的?

4.5 什么是 ISA 总线?它在现代微型计算机中是通过什么部件与系统连接的?

- 4.6 什么是 PCI 局部总线？它通过什么部件与 CPU 总线连接？
- 4.7 ISA 总线能插入几位的接口卡？PCI 总线呢？
- 4.8 什么是 AGP 总线？它的主要用途是什么？
- 4.9 哪种总线支持即插即用？微型计算机系统要实现即插即用应满足哪些条件？
- 4.10 USB 能做什么？请列举出你所知道的设备中哪些使用了 USB 总线。
- 4.11 USB 设备需要外接电源吗？它的电源供应是从哪里获取的？
- 4.12 USB 1.1 所规定的最大速率是多少？USB 2.0 的最大速率是多少？
- 4.13 USB 的拓扑结构是_____？
- A. 总线型结构 B. 星形结构 C. 环形结构 D. 点对点结构
- 4.14 通过_____可以无需计算机主机就能实现两台外设间的数据传输。
- A. USB B. IEEE 1394 C. PCI - Express D. RS - 232
- 4.15 以下说法中，哪些是错误的？
- A. USB 与 IEEE 1394 都支持即插即用；
- B. 因为 USB 支持即插即用，所以在 Windows 98、Windows Me、Windows 2000、Windows XP 系统中使用 USB 设备无需手工安装驱动程序；
- C. USB 的任何传输过程都必须由主机发起，设备只能被动响应；
- D. USB 可以支持多达 127 个物理设备；
- E. USB 可以同时支持许多具有不同数据速率的设备；
- F. 当 USB 连线长度不够时，允许用 USB 延长线或 USB Hub，使连接距离扩充到任意长度。
- 4.16 设计一个 ISA 总线接口，使它包含一个 I/O 地址为 0800H 的 8 位输入端口和一个 I/O 地址为 0A00H 的 8 位输出端口（要求用 74LS244 三态缓冲器和 74LS373 锁存器实现）。

第5章 指令系统

本章在简要介绍微型计算机指令一般格式的基础上,较为详细地叙述了 8086 指令系统中各类指令的功能和寻址方式,并进一步介绍了 80X86 CPU 新增指令的功能。

5.1 指令系统概述

控制计算机完成各种指定操作的命令称为指令。一台计算机所能执行的各种指令的集合称为它的指令系统。不同的计算机具有各自不同的指令系统。一个指令系统的功能是否强大与计算机的性能有很大的关系。指令系统的设置与机器的硬件系统结构是密切相关的,指令系统的功能越齐全、通用性越强、指令越丰富,就越需要复杂的硬件结构来支持。

早期的微型计算机因受集成电路技术水平的限制,其硬件结构比较简单,所支持的指令系统一般只有定点数的加减运算、数据传送、转移等几十条最基本的指令。后来,随着集成电路、特别是超大规模集成电路(VLSI)技术的发展,硬件结构变得越来越复杂,功能也不断增强,指令系统也就变得越来越丰富了。在现代计算机的指令系统中,除以上基本指令外,增加了乘除运算、浮点运算、十进制运算、条件转移、子程序调用及字符串处理等指令,另外,为了便于操作系统的实现和优化,还设置了控制系统状态的特权指令、管理多道程序和多处理机系统的专用指令等。

丰富的指令系统大大提高了计算机的性能,但也带来一些负面因素。如因指令系统过于庞大而使设计周期变长、维护困难等。事实上,最经常使用的指令还是数据传送、算术和逻辑运算、转移、子程序调用等几十条最基本的指令。

目前,硬件技术的发展速度已远远超过了软件技术的发展,软件的成本在系统中变得越来越高。为了在不断发展的计算机上能够继承已有的软件以降低软件费用,人们提出了软件“兼容”的概念。这样,在 20 世纪 60 年代就出现了系列(Series)计算机。所谓系列计算机是指基本指令系统相同、基本体系结构相同的一系列计算机,如 IBM PC(XT/AT/286/386/486/Pentium)微型计算机系列等。系列机虽然在结构和性能上不断地完善和提高,但其基本结构体系没有变,指令系统向上兼容,即新机种上的指令系统包含了能够在旧机种上运行的全部指令。

基于以上两点,本章主要讨论 8086CPU 的六大类常用基本指令。

5.1.1 指令的格式

指令的格式与机器的字长、存储器的容量及指令的性质都有很大的关系。计算机是通过执行指令来完成各种任务的,微处理器处理的所有对象除指令外都可以认为是数据,所以计算机所要完成的任务就是对各种数据的处理。因此,一条指令至少要包括这样几个信息:运算数据的来源、运算结果的去向及执行的操作。

指令要执行的操作称为操作码,也称为指令码。它说明所执行的操作的性质和功能。操作码有的占用一个字节,有的占用两个字节。一台计算机有几十或几百条指令,每条指令都有一个对应的操作码,计算机通过识别操作码来决定要进行的操作。

参加运算的数据,亦即操作的对象称为操作数。数据的来源及运算结果的去向都统称为操作数的地址,也可简称为操作数。通常将存放结果的操作数称为目标操作数,相应的另一个操作数就称为源操作数。

除操作码和操作数外,在遇到转移、子程序调用等非顺序执行的情况时,指令中还应给出下一条要执行的指令的地址(在程序顺序执行时,下条要执行的指令的地址由指令指针IP指出)。

总之,一条指令主要包括了两种信息,即操作码和操作数(或称地址码)。操作码(Operation Code)部分描述操作的性质(如加、减、乘、除等),操作数(Operation Data)描述了操作的对象,它可以是直接参加运算的数据,也可以是数据的地址。如何确定操作数的地址,就是下一节要讨论的寻址方式。

指令的长度与操作码和操作数的多少及其类型都有关系。操作数越多,其指令的长度就越长。一条8086指令的长度一般在1~7个字节之间。第一个字节通常为指令的操作码。

1. 零操作数指令

格式:

OP

这里,OP为指令操作码。这种指令格式中只有操作码而没有操作数。这种指令一般有两种:一是不需要任何操作数。如空操作指令、暂停指令等;二是操作数是隐含的,即指令有默认的操作数。如字符串操作指令:MOVSB。该指令表示将源地址中的字节数传送到目标地址中去。

2. 单操作数指令

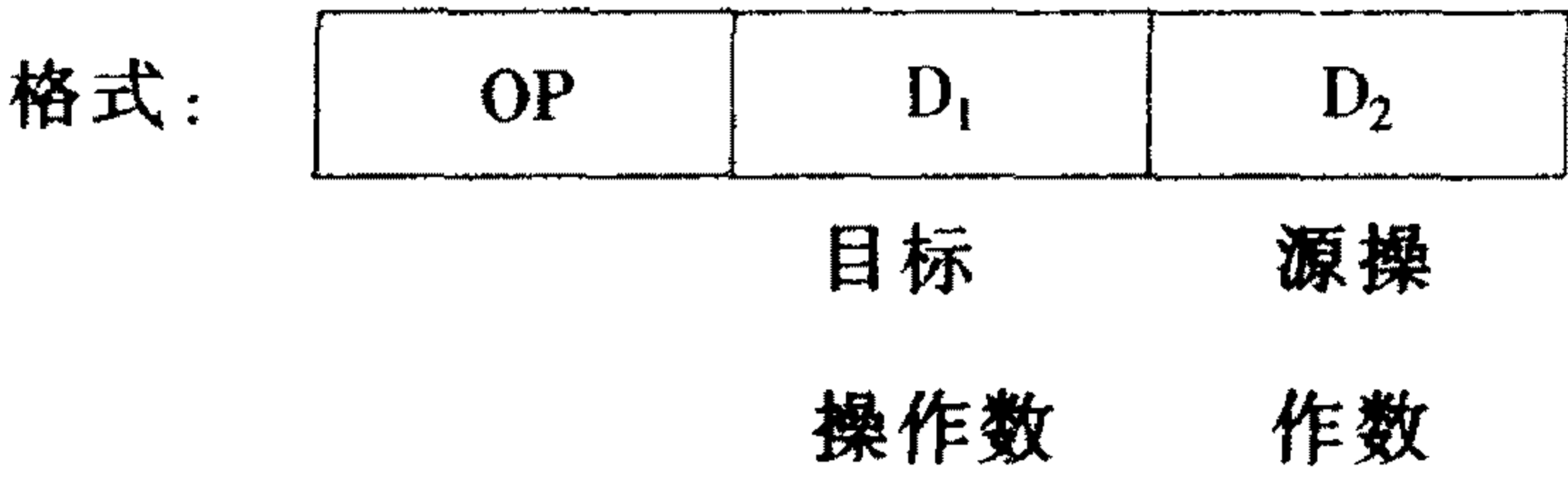
格式:

OP	D ₁
----	----------------

这里得 D_1 表示操作数,它有两种含义:

- ① 一是该操作数既表示运算数据的地址,也表示存放运算结果的地址。如加 1 指令: INC AX。该指令执行的结果是将累加器 AX 中的内容加 1 然后再送回 AX;
- ② 指令中的操作数 D_1 是源操作数,而目标操作数被隐含了,即指令实际上是双操作数指令,默认了另一个操作数,运算的结果也默认存放在隐含的操作数地址中。如乘法指令: MUL BL,该指令的执行是将累加器 AL 的内容与 BL 内容相乘,结果送 AX。

3. 双操作数指令



指令中的 D_2 是源操作数,它本身就是参加运算的一个数据,或是其中一个操作数的地址; D_1 是目标操作数,它有两个含义,表示另一个操作数的地址及存放运算结果的目标地址。这是最常见的指令格式,微处理器的基本指令系统中多数指令都是双操作数指令。例如:加法指令 ADD AX,BX。指令中的源操作数是 BX,表示参加运算的操作数之一是 BX 的内容,AX 为目标操作数,它既表示参加运算的另一数据是 AX 的内容,同时也表示运算的结果存放在 AX 中。该指令执行的结果是将 AX 的内容加上 BX 的内容,结果送 AX 中。

微型计算机中的指令都是以上三种格式。在大型机或某些功能较强的中、小型机中,常设置有一些功能很强、用于处理成批数据的指令,如字符串处理、向量及矩阵运算等。在这些指令中,有时需要用多个操作数来描述数据存放的首地址、数据长度、下标等信息。这一类的指令格式就不再限于以上三种,而可以是三操作数以至多操作数的指令格式。

在计算机中,无论是指令码还是数据都是以二进制码的形式存放在存储器中的,仅从表面上看不出二者有什么区别。但指令的地址是由指令指针 IP 确定,数据的地址则是由指令中给出,CPU 在访问存储器时是绝对不会将指令和数据混淆的。

5.1.2 指令中的操作数

8086 指令系统中的指令均为以上三种格式。其操作数既可以是数据,也可以是参加运算的数据的地址。根据它们的性质,将操作数分为这样三类:立即数操作数、寄存器操作数和存储器操作数。

1. 立即数操作数

所谓立即数是指具有固定数值的操作数,即常数。它不会由于指令的执行而发生变化。

它可以是字节(8 位)或字(16 位),当它们分别代表无符号数和有符号数时,其各自的取值范围见表 5-1。

表 5-1 立即数操作数的取值范围

	8 位数	16 位数
无符号数	00H ~ 0FFH(0 ~ 255)	0000H ~ 0FFFFH(0 ~ 65535)
带符号数	80H ~ 7FH(- 128 ~ + 127)	8000H ~ 7FFFH(- 32768 ~ + 32767)

如果一个立即数的取值超出了规定的范围,就会发生错误。在指令中,立即数操作数表示参加运算的数据而不是数据的地址,所以只能用作源操作数,而不能作为目标操作数。

2. 寄存器操作数

寄存器操作数是 8086 的 8 个通用寄存器或段寄存器,表示数据的地址或运算结果的地址。它既可以用做源操作数,也可以用做目标操作数。

通用寄存器主要用于存放操作的数据。通用寄存器中的 AX、BX、CX、DX 既可以作为 4 个 16 位寄存器,用来存放字操作数,也可以当作 8 个 8 位寄存器(AH、AL、BH、BL、CH、CL、DH、DL),用来存放字节操作数。SI、DI、BP、SP 只能存放字操作数。

段寄存器用来存放当前操作数据的段基地址。在与通用寄存器或存储器传送数据时,段寄存器可作为源操作数或目标操作数(但代码段寄存器 CS 一般不作为目标操作数)。此外,8086 不允许用一条指令将立即数传送到段寄存器。如果需要这样做,可用某个通用寄存器作为中间桥梁,用两条传送指令实现。

仅有个别指令将标志寄存器 FLAGS 作为指令的操作数。

3. 存储器操作数

存储器操作数表示当前操作数据的地址或运算结果的地址。故在指令中既可作为源操作数,也可以作为目标操作数。操作的数据可以是字节、字或双字,分别存放在 1 个、2 个或 4 个存储单元中。但在 8086 中,大多数指令不允许源操作数和目标操作数同时为存储器操作数,也就是说,不允许从存储器到存储器的操作。若有这样的需要,可以先将其中一个存储器的内容传送到某个通用寄存器中,然后再把这个寄存器与另一个存储器的内容作为操作数执行希望的操作。

从第 3 章中已经知道,能够惟一标识一个存储器单元的是它的物理地址,而物理地址是由段基地址和偏移地址两部分构成。所以,要寻找一个存储在存储器中的操作数,必须首先确定数据所在的段,即确定有关的段寄存器。一般情况下,若指令中没有指明所涉及的段寄存器,则 CPU 就采用默认的段寄存器来确定数据所在的段。各种存储器操作所约定的默认

段寄存器、段超越(即显式地指明段寄存器)所允许的段寄存器以及指令的有效地址所在的寄存器见表 5-2。

数据的偏移地址可以通过不同的寻址方式由指令给出。在下一节的“寻址方式”中,将介绍寻找操作数地址的各种方法。

表 5-2 段寄存器使用的一些基本约定

存储器操作类型	默认的段寄存器	允许超越的段寄存器	段内偏移地址来源
取指令	CS	无	IP
堆栈操作	SS	无	SP
通用数据读/写	DS	CS, ES, SS	按寻址方式取得
串操作源串地址	DS	CS, ES, SS	SI
串操作目标串地址	ES	无	DI
BP 作为基址寄存器	SS	CS, DS, ES	按寻址方式取得

5.1.3 指令的字长及执行时间

1. 指令字长与机器字长

机器字长是指计算机能够直接处理的二进制数的位数,是计算机的一个重要技术指标,它决定了计算机的运算精度,字长越长,运算的精度就越高。因主存储器都是按字节编址的,每个地址单元存放 8 位二进制码,为尽可能充分利用存储空间并便于数据处理,机器的字长都设计成字节长度的 1、2、4、或 8 倍,即 8 位、16 位、32 位或者 64 位,对微型计算机来讲,目前其字长一般都是 32 位,即计算机能够直接处理 32 位二进制数。

指令的字长包括操作码的长度、操作数地址的长度及操作数的个数。在微型计算机中,操作码的长度是不固定的(操作码采用可变格式的编码格式),操作数的格式也不同,所以指令的长度也是不固定的。为了充分利用存储空间,指令长度一般为字节的整数倍。例如:8086CPU 的指令长度为 1~7 个字节,80286 指令中的最大长度是 10 个字节,80386 和 80486 指令的最大长度为 15 个字节。

8086CPU 的指令系统采用变字长的指令格式。指令中包括操作码、寻址方式以及操作数三部分信息,其指令编码的一般形式如图 5.1 所示。指令第一个字节是操作码,说明操作的性质;第二个字节规定操作数的寻址方式,包括 mod(方式)、reg(寄存器)和 reg/mem(寄存器/存储器)三个域,用于操作数的类型及寻找它的方法。其中:

① reg 域用来规定一个操作数为寄存器操作数,至于它是源操作数还是目标操作数则由操作码确定;

② mod 域用来确定另一个操作数是寄存器操作数还是存储器操作数,并在必要时规定后面的位移量长度;

③ 当 mod 域确定另一个操作数是寄存器操作数时,它负责给出寄存器号。若是存储器操作数则指出如何求得存储单元的有效地址。

最后的字节是操作数,它的长度由 mod 域确定。

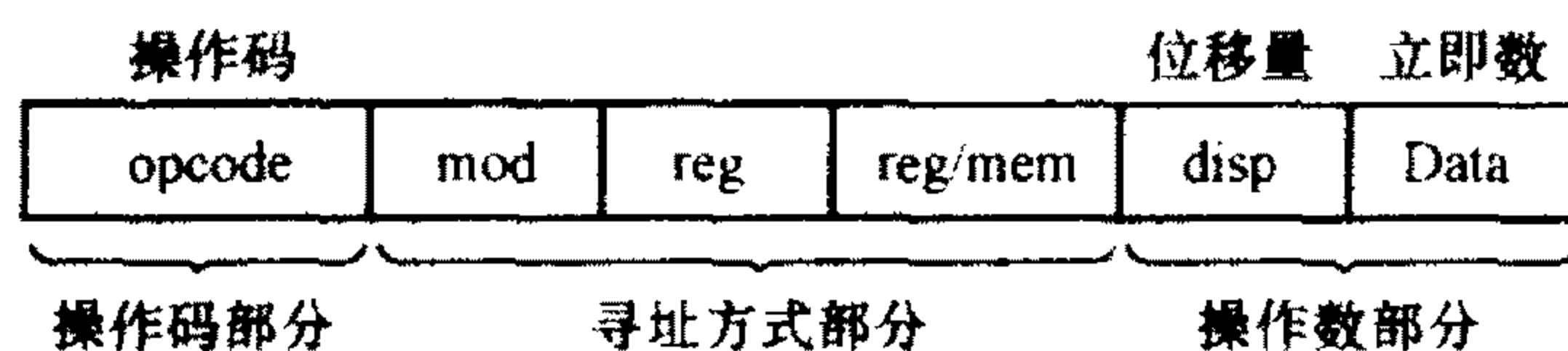


图 5.1 8086 指令编码的一般格式

指令的长度可以小于或等于机器的字长,也可以大于机器字长。前者称为短格式指令,所占的存储空间比较小,执行速度快,一般都将最常用的指令(如数据传送、算术运算等)设计成短格式。对于操作复杂的指令,因所需的信息量大,则设计为长格式。

2. 指令的执行时间

了解指令的执行时间有时是很重要的。例如在用软件实现定时或延时时,需要估算出一段程序的运行时间。另外,在某些实时控制要求较严或对程序运行时间要求较高的场合,除需认真研究程序的算法外,对选择什么样的指令及采用什么样的寻址方式也是很重要的。因为不同的指令在执行时间上有很大的差别,而不同的寻址方式其计算偏移地址所需时间也不同。由于指令的种类很多,要详细讨论各种指令的执行时间比较困难,这里仅以 8088CPU 指令系统为例,对指令的执行时间作一般的讨论。

一条指令的执行时间应包括取指令、取操作数、执行指令及传送结果几个部分,单位用时钟周期数表示。

不同指令的执行时间有较大的差别,表 5-3 列出了部分常用指令的执行时间及其访问存储器的次数。

存取操作数的时间与采用的寻址方式有关。寄存器操作数占用的时间最短,而对存储器操作数则还需考虑计算偏移地址所花的时间。表 5-4 为不同寻址方式下,计算偏移地址所需要的时间。

表 5-3 8088CPU 常用指令执行时间

指 令		所需时钟周期数	访问内存次数
MOV	累加器到内存	10(14)	1
	内存到累加器	10(14)	1
	寄存器到寄存器	2	0
	内存到寄存器	8(12) + EA	1
	寄存器到内存	9(13) + EA	1
MOV	立即数到寄存器	4	0
	立即数到内存	10(14) + EA	1
	寄存器到段寄存器	2	0
	内存到段寄存器	8(12) + EA	1
	段寄存器到寄存器	2	0
	段寄存器到内存	9(13) + EA	1
ADD 或 SUB	寄存器到寄存器	3	0
	内存到寄存器	9(13) + EA	1
	寄存器到内存	16(24) + EA	2
	立即数到寄存器	4	0
	立即数到内存	17(25) + EA	2
MUL	累加器乘 8 位寄存器	70 ~ 77	0
	累加器乘 16 位寄存器	118 ~ 133	0
	累加器和内存字节乘	(76 ~ 83) + EA	1
	累加器和内存字乘	[124(128) ~ 139(143)] + EA	1
IMUL	累加器乘 8 位寄存器	80 ~ 98	0
	累加器乘 16 位寄存器	128 ~ 154	0
	累加器和内存字节乘	(86 ~ 104) + EA	1
	累加器和内存字乘	[134(138) ~ 160(164)] + EA	1
DIV	除数在 8 位寄存器中	80 ~ 90	0
	除数在 16 位寄存器中	144 ~ 162	0
	除数为 8 位内存数	(86 ~ 96) + EA	1
	除数为 16 位内存数	[150(154) ~ 168(172)] + EA	1
IDIV	除数在 8 位寄存器中	101 ~ 112	0
	除数在 16 位寄存器中	165 ~ 184	0
	除数为 8 位内存数	(107 ~ 118) + EA	1
	除数为 16 位内存数	[171(175) ~ 190(194)] + EA	1

续表			
指 令		所需时钟周期数	访问内存次数
循环和移位	在寄存器中移 1 位	2	0
	在寄存器中移若干位	$8 + 4 \times \text{位数}$	0
	内存数据移 1 位	$15(23) + \text{EA}$	2
	内存数据移若干位	$20(28) + \text{EA} + 4 \times \text{位数}$	2
JMP	段内/段间直接转移	15	0
	段内间接转移	$8(12) + \text{EA}$	1
	段间间接转移	$24(32) + \text{EA}$	2
条件转移	JCXZ	6(不转移)	0
		18(转移)	0
	其他条件转移指令	4(不转移)	0
		16(转移)	0

注：① 表中 EA 表示偏移地址。小括号内的数为 8088 进行字操作的时钟数。因为 8088 的数据线只有 8 位，每个总线周期只能传送一个字节，所以对字操作要再加上 4 个时钟周期。

② 对条件转移指令，若条件满足，执行的时间比较长。因为要产生转移，就要包括取下一条指令所需的时间。若条件不满足，执行时间就较短。因为此时不产生转移，而是执行下一条指令。

表 5-4 计算偏移地址 EA 所需时间

寻址方式		计算 EA 所需时钟数
直接寻址		6
寄存器间接寻址		5
寄存器相对寻址		9
基址、变址寻址	$[\text{BX} + \text{SI}], [\text{BX} + \text{DI}]$	7
	$[\text{BP} + \text{SI}], [\text{BP} + \text{DI}]$	8
基址、变址加相对寻址	$[\text{BX} + \text{SI} + \text{位移量}], [\text{BP} + \text{DI} + \text{位移量}]$	11
	$[\text{BX} + \text{DI} + \text{位移量}], [\text{BP} + \text{SI} + \text{位移量}]$	12

注：① 若有段超越，则需再加上两个时钟周期。

② 寻址方式的介绍参见 5.2 节。

从上面的表中可以看到，对同一种指令，如果寻址方式不同，其指令执行时间可能相差很大。在以上讨论的三种类型的操作数中，寄存器操作数的指令执行速度最快，立即数操作数次之，存储器操作数指令的执行速度最慢。这是由于寄存器位于 CPU 的内部，执行寄存器操作数指令时，8086 的执行单元(EU)可以简捷地从 CPU 内部的寄存器中取得操作数，不需要访问内存，因此执行速度很快。立即数操作数作为指令的一部分，在取指时被 8086 总线接口单元(BIU)取出后存放在 BIU 的指令队列中，执行指令时也不需要访问内存，因而执

行速度也比较快。而存储器操作数放在某些内存单元中,为了取得操作数,首先要由总线接口单元计算出其所在单元的 20 位物理地址,然后再执行存储器的读/写操作。所以相对前述两种操作数来说,指令的执行速度最慢。

以通用数据传送指令(MOV)为例,若 CPU 的时钟频率为 5 MHz,即一个时钟周期为 $0.2\ \mu\text{s}$,则从寄存器到寄存器之间的传送指令的执行时间为

$$t = 2 \times 0.2 = 0.4\ \mu\text{s}$$

立即数传送到寄存器的指令执行时间为

$$t = 4 \times 0.2 = 0.8\ \mu\text{s}$$

而存储器到寄存器的字节传送,设存储器采用基址、变址寻址方式,则指令执行时间为

$$t = (8 + \text{EA}) \times 0.2 = (8 + 8) \times 0.2 = 3.2\ \mu\text{s}$$

5.2 寻址方式

所谓寻址方式,是指寻找本条指令中数据的地址或是指定在转移类指令中确定转移的目标地址的方法。前者称为寻找操作数的寻址方式,后者称为寻找指令地址的寻址方式。寻址方式是指令系统中一个很重要的内容。它与计算机的硬件结构密切相关,对指令的格式和功能都有很大的影响。在汇编语言的程序设计及高级语言的编译程序设计中,都需要对计算机的寻址方式非常了解。

对不同系列的微处理器,其寻址方式不完全一样,但它们基本的工作原理是一样的。本节所讨论的寻址方式是以 8086CPU 为例,由于 8086 指令系统与 80x86 及 Pentium 系列微处理器指令系统兼容,因此,这里所讲述的内容也适用于 Intel 公司系列微处理器,其改进部分将在本章的最后一节介绍。

5.2.1 寻找操作数的寻址方式

在程序的执行过程中,操作数可能存放在寄存器中,也可能存放在主存储器或 I/O 端口中,或者由指令直接给出(此时的操作数事实上也在存储器中)。对于指令中直接给出的立即数操作数或寄存器操作数,其寻址方法是很简单的,比较复杂的是寻找存放在存储器中的操作数。下面讨论 8086CPU 的几种基本寻址方法。

1. 立即寻址

立即寻址(Immediate Addressing)是指指令中的源操作数是参加操作的一个立即数,由指令直接给出。它作为指令的一部分,紧跟在指令的操作码之后,存放于内存的代码段中,在

CPU 取指令时随指令码一起取出并直接参加运算。

这种寻址方式减少了 CPU 访问存储器的次数,使指令的执行速度提高。但由于立即数是指令的一部分,在程序运行中不能改动,而多数情况下指令处理的数据是在不断变化的,如上条指令执行的结果可能是下条指令处理的数据。所以立即寻址方式使用的范围很窄,主要用于给寄存器或存储单元赋初值。

指令中的立即数可以是 8 位或 16 位的整数。若为 16 位,则存放时低 8 位存放在低地址单元,高 8 位存放在高地址单元。如图 5.2 所示。

例 5-1 指令“MOV AX,2233H”表示将 16 位的立即数 2233H 送入累加器 AX。指令执行后, AH = 22H, AL = 33H。

这是一条三字节指令,图 5.2 给出了指令执行情况示意图。

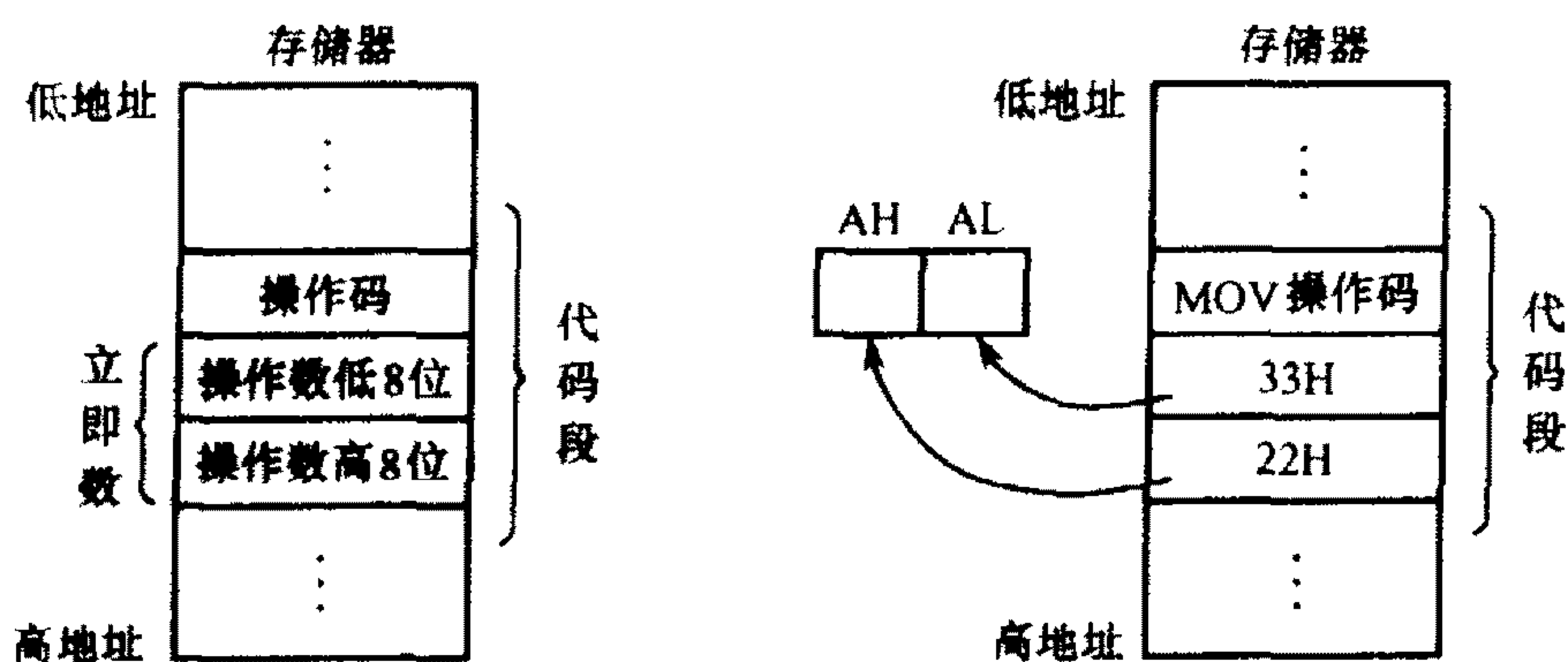


图 5.2 立即寻址操作示意图

2. 直接寻址

直接寻址 (Direct Addressing) 指令是在指令中直接给出操作数的 16 位偏移地址,该地址与指令的操作码一起存放在内存的代码段,也是低 8 位在低地址,高 8 位在高地址。但是,若操作数本身在指令中无特殊声明,默认存放在内存的数据段中。

例 5-2 指令“MOV AX,[2233H]”将数据段中偏移地址为 2233H 和 2234H 两单元的内容送到 AX 中。

这时,如果 (DS) = 2000H,则所寻找的操作数的物理地址为

$$20000H + 2233H = 22233H$$

指令的执行情况如图 5.3 所示。

要注意区别直接寻址指令与前面介绍的立即寻址指令二者的不同。直接寻址指令中的数值是操作数的 16 位偏移地址,而不是操作数本身。为了区分二者,指令系统规定偏移地址必须用方括号括起来。

在上例中,指令的执行不是将立即数 2233H 送到累加器 AX,而是将偏移地址为 2233H 的内存单元中的内容送 AX。若操作数不是存放在 DS 段,则在指令中要用段超越符号加以声明。

例 5-3 指令“MOV BX,ES:[1200H]”将附加段中偏移地址为 1200H 和 1201H 两单元的内容送到 BX 寄存器中。

在汇编语言中,有时也用一个符号来代替数值以表示操作数的偏移地址,通常把这个符号称为符号地址。上例中,若用 BUFFER 代替偏移地址 1200H,则指令可写成

MOV BX,ES:[BUFFER]

这两者是等效的,但 BUFFER 必须在程序的起始处予以声明。

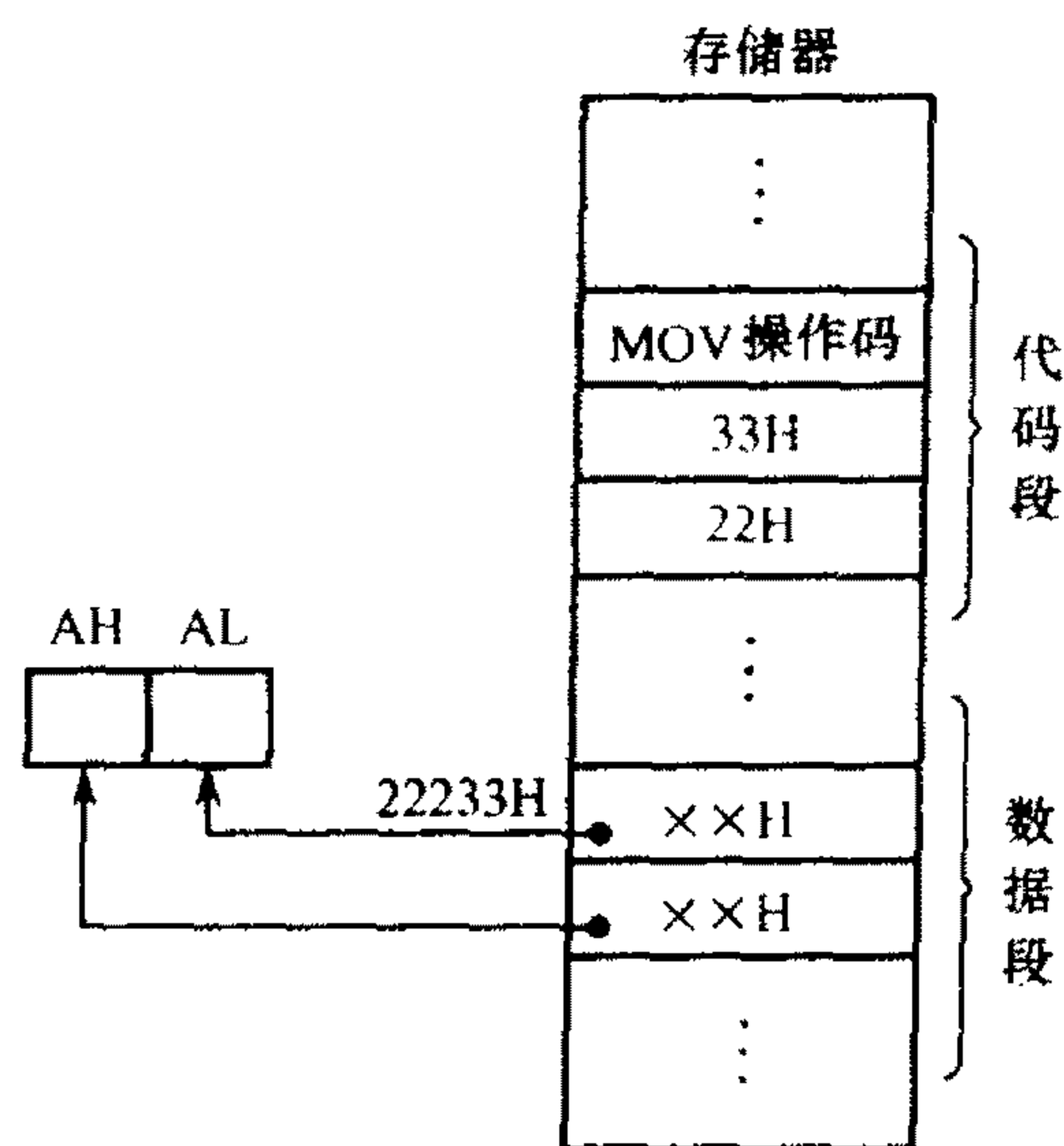


图 5.3 直接寻址方式

3. 寄存器寻址

寄存器寻址(Register Addressing)中指令的操作数为 CPU 的内部寄存器。它们可以是数据寄存器(8 位或 16 位),也可以是地址指针、变址寄存器或段寄存器。

例 5-4 指令“MOV SI,AX”将 AX 的内容送到寄存器 SI 中,其执行情况如图 5.4 所示。若指令执行前 (AX) = 2233H, (SI) = 4455H,则指令执行后 (SI) = 2233H,而 (AX) 中的内容保持不变。

采用寄存器寻址时,虽然指令的操作码在代码段中,但操作数在 CPU 的寄存器中,指令在执行时不必通过访问内存就可取得操作数,故执行速度较快。

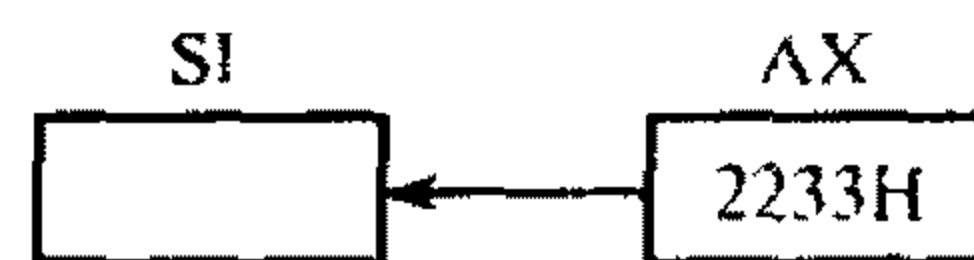


图 5.4 寄存器寻址示意图

4. 寄存器间接寻址

寄存器间接寻址(Register Indirect Addressing)与寄存器寻址方式不同,指令中指定的寄存器的内容不是操作数,而是操作数的偏移地址。也就是说操作数的偏移地址放在寄存器中,操作数本身则在存储器中。存放存储器地址的寄存器有时也称为地址指针。寄存器间接寻址方式可用的寄存器只允许是 SI、DI、BX 和 BP 4 个,它们可简称为间址寄存器。选择不同的间址寄存器,涉及的段寄存器将有所不同。在默认情况下,选择 SI、DI、BX 为间址寄存器时,操作数在数据段,段基地址由 DS 决定;选择 BP 为间址寄存器,则操作数在堆栈段,段基地址由 SS 决定。但无论选择哪一个间址寄存器,都允许段超越,即可在指令中用段超越前缀指明当前操作数在哪一个段。

因为间址寄存器中存放的是操作数地址,所以根据规定,用作间址的寄存器必须加上方

括号,以避免与寄存器寻址指令混淆。

例 5-5 指令“MOV AX,[SI]”的寻址过程如图 5.5 所示。

已知(DS) = 6000H, (SI) = 1200H。因为指令中没有特别的声明(指定段超越),所以操作数默认在数据段。可计算出操作数的物理地址为 $60000H + 1200H = 61200H$

执行的结果为

$$(AX) = 3344H$$

若操作数是在附加段,则本例中的指令应表示成以下形式:

MOV AX,ES:[SI]

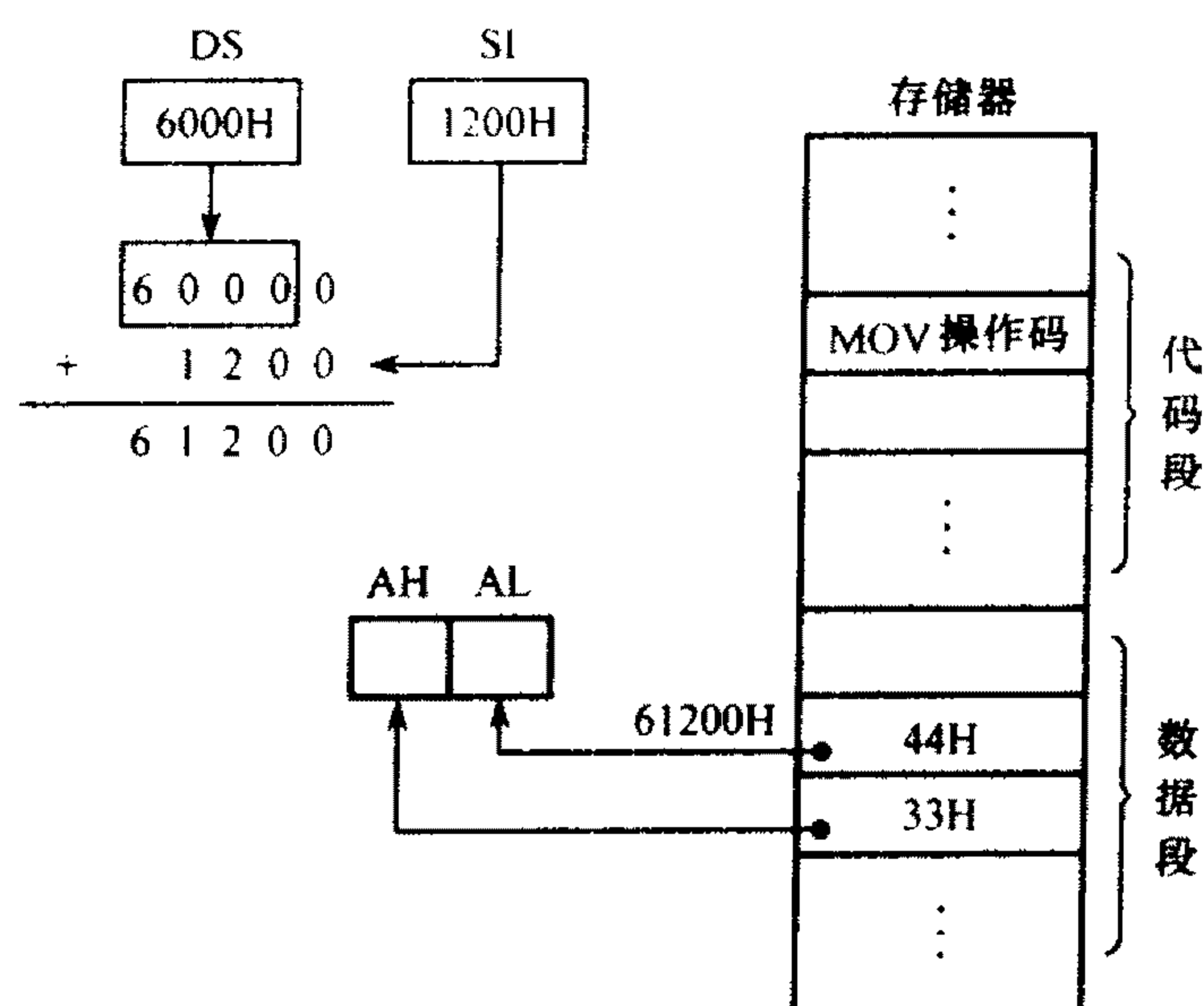


图 5.5 寄存器间接寻址执行示意图

例 5-6 在指令“MOV BX,[BP]”中,由于用 BP 作间址寄存器,故操作数默认在堆栈段(SS 段)中。若已知(SS) = 8000H, (BP) = 0200H,则操作数的物理地址为

$$80000H + 0200H = 80200H$$

指令执行结果为: BL = [80200H]中的内容, BH = [80201H]的内容。

由于 4 个间址寄存器包括两个基址寄存器 BX、BP 以及两个变址寄存器 SI、DI,因此有些书上又将寄存器间接寻址方式分为基址寻址和变址寻址。基址寻址方式使用 BX、BP 作为间址寄存器,而变址寻址方式使用 SI 和 DI 作间址寄存器。

5. 寄存器相对寻址

采用这种寻址方式时,存放于主存中的操作数的偏移地址等于间址寄存器的内容加上指令中给出的一个 8 位或 16 位的地址位移量,位移量紧随操作码一起存放在代码段中。操作数默认在哪个段,则仍由所使用的间址寄存器决定(使用 BP 则默认在堆栈段,其他的

则默认在数据段) 位移量也可看作相对值,故把这种带位移量的寄存器间接寻址方式称为寄存器相对寻址。

例 5-7 指令“MOV AX, DATA[BX]”的寻址过程示例。

设: (DS) = 6000H, (BX) = 1000H, DATA = 08H, 则

$$\text{物理地址} = 6000\text{H} + 1000\text{H} + 08\text{H} = 61008\text{H}$$

指令的执行情况如图 5.6 所示。执行结果为

$$(\text{AX}) = 5566\text{H}$$

寄存器相对寻址常用于存取表格或一维数组中的元素——把表格的起始地址作为位移量,元素的下标值放在间址寄存器中(反过来也可以),即可存取表格中的任意一个元素。

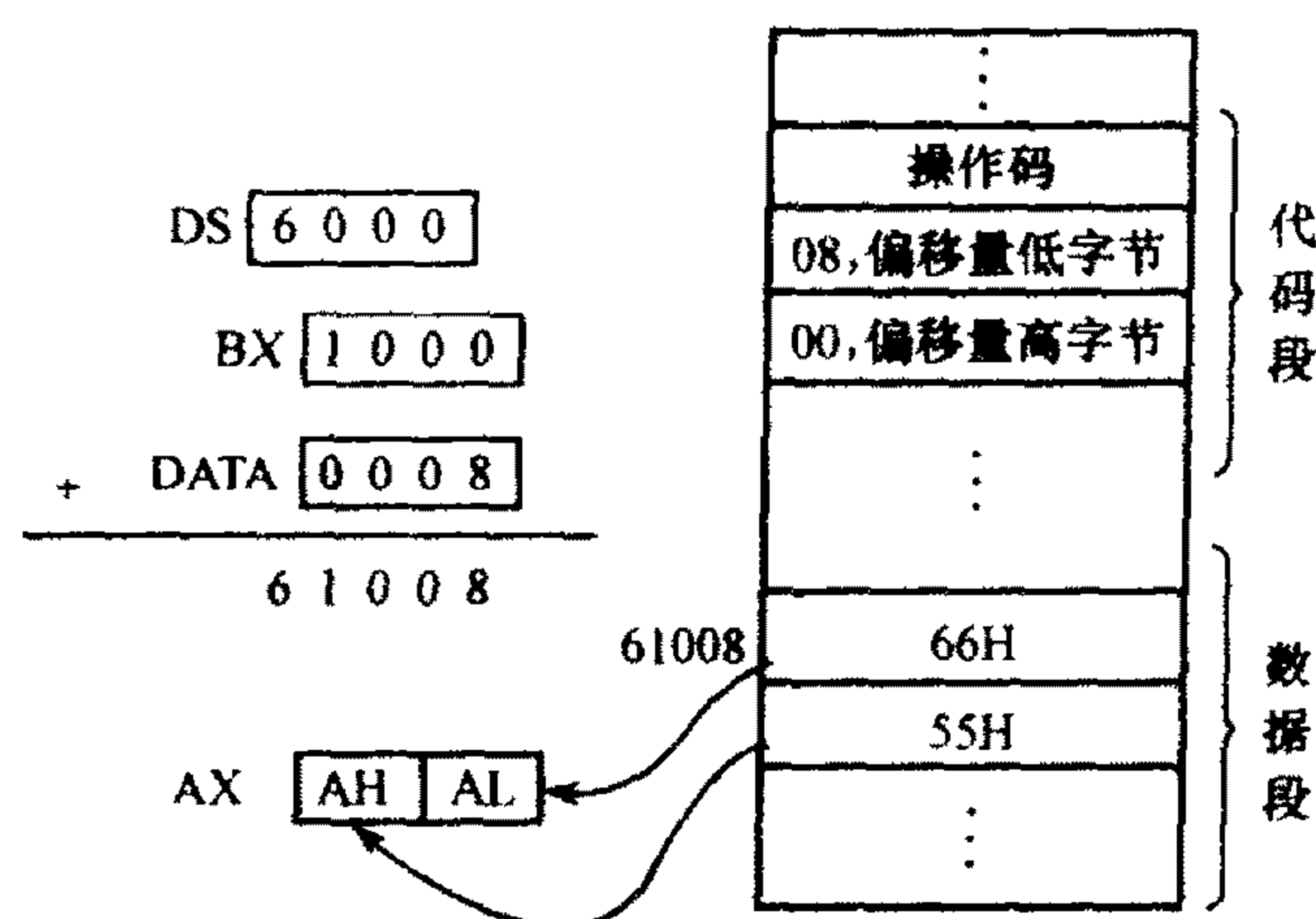


图 5.6 寄存器相对寻址示意图

在汇编语言中,相对寻址指令的书写格式允许有几种不同的形式。以下几种写法是完全等价的:

```
MOV AL, DATA[SI]
MOV AL, [SI]DATA
MOV AL, DATA + [SI]
MOV AL, [SI] + DATA
MOV AL, [DATA + SI]
MOV AL, [SI + DATA]
```

6. 基址 - 变址寻址

这种寻址方式由一个基址寄存器(BX 或 BP)的内容和一个变址寄存器(SI 或 DI)的内容相加而形成操作数在主存中的偏移地址。数据的段基地址在默认情况下则仍由指令中使用的基址寄存器决定,即若使用 BX 作基址寄存器,则数据默认在数据段;若使用 BP 作基址寄

寄存器,则数据默认在堆栈段。但允许段重设。

例 5-8 指令 $\text{MOV AX}, [\text{BX}][\text{SI}]$ 的寻址过程示例。

设: $(\text{DS}) = 8000\text{H}$, $(\text{BX}) = 2000\text{H}$, $(\text{SI}) = 1000\text{H}$, 则

物理地址 $= 8000\text{H} + 2000\text{H} + 1000\text{H} = 83000\text{H}$

指令执行后:

$(\text{AL}) = [83000\text{H}]$, $(\text{AH}) = [83001\text{H}]$

指令执行情况如图 5.7 所示。

使用基址 - 变址寻址方式时, 8086CPU 不允许将两个基址寄存器或两个变址寄存器组合在一起寻址, 即指令中不允许同时出现两个基址寄存器或两个变址寄存器。例如, 以下指令是非法的:

$\text{MOV AX}, [\text{BX}][\text{BP}]$; 错误! 同时出现两个基址寄存器

$\text{MOV AX}, [\text{SI}][\text{DI}]$; 错误! 同时出现两个变址寄存器

7. 基址 - 变址 - 相对寻址

这种寻址方式事实上是上一种方式的扩充。指令中指定了一个基址寄存器和一个变址寄存器, 同时还给出一个 8 位或 16 位的位移量, 将三者相加就得到操作数在主存中的偏移地址。至于默认的段寄存器, 仍由所用的基址寄存器决定。指令同样允许段重设。

例 5-9 指令“ $\text{MOV AX}, \text{DATA}[\text{DI}][\text{BX}]$ ”的寻址过程示例。

该指令将段地址为 (DS) , 而偏移地址为 $(\text{BX}) + (\text{DI}) + \text{DATA}$ 的连续两个存储单元的内容送到 AX 。指令的执行情况如图 5.8 所示。

使用这种寻址方式可以很方便地访问二维数组。例如用基址寄存器存放数组的首地址(偏移地址), 而变址寄存器和位移量分别存放行和列的值, 则利用基址 - 变址 - 相对寻址指令就可以直接访问二维数组中指定的行和列的元素。

与寄存器间接寻址方式类似, 基址 - 变址 - 相对寻址指令同样也可以表示成多种

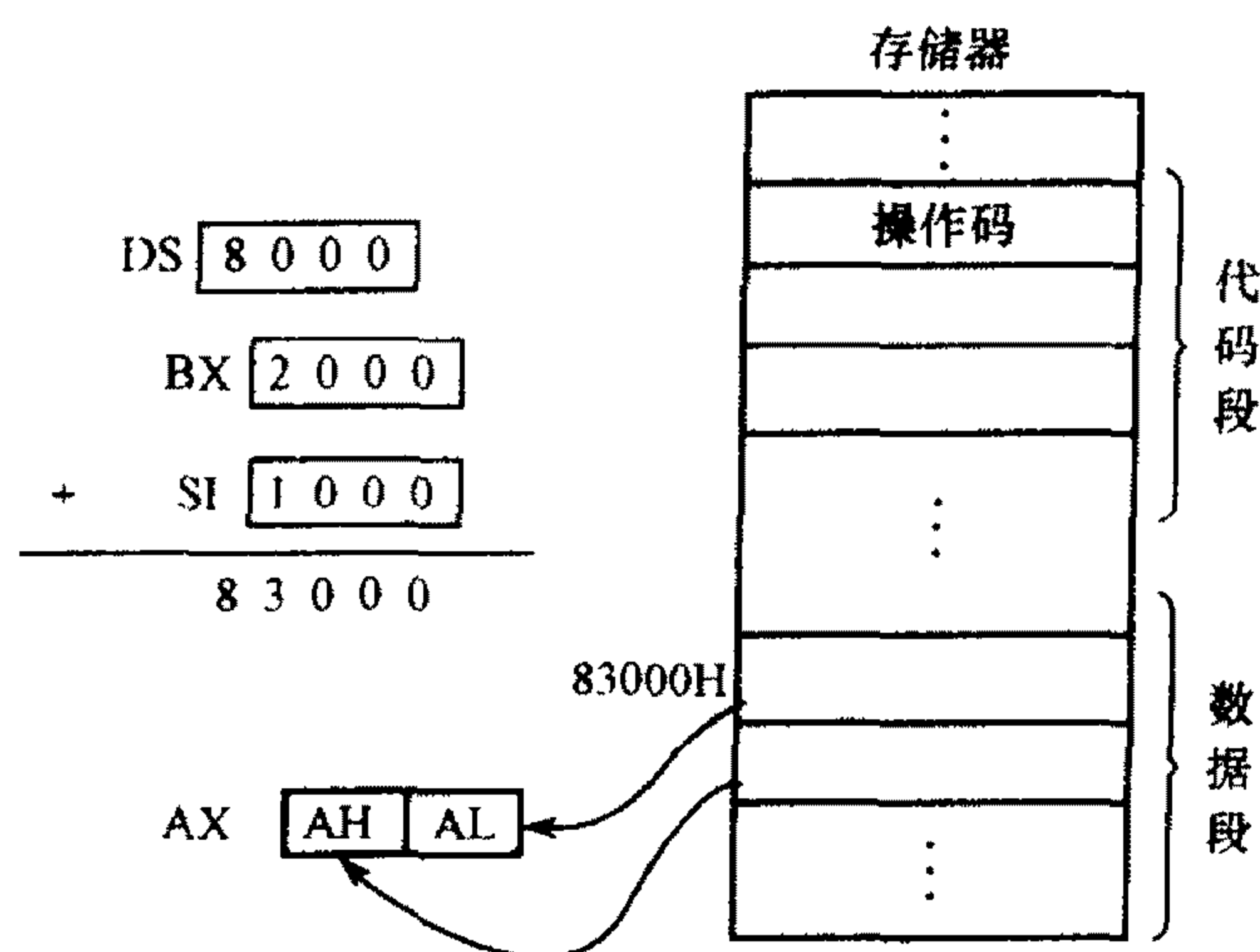


图 5.7 基址 - 变址寻址

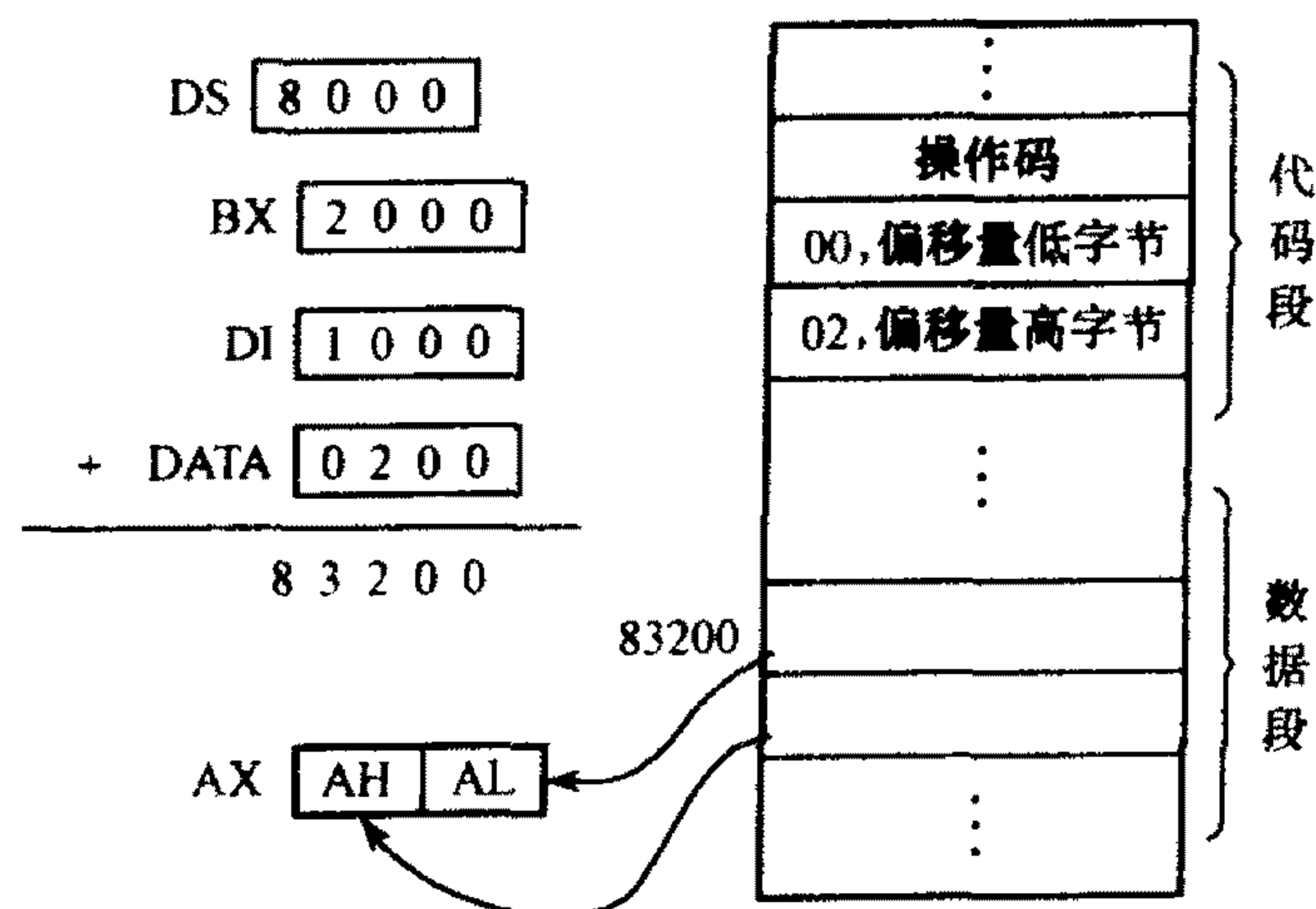


图 5.8 基址 - 变址 - 相对寻址示意图

形式,例如:

```
MOV AX, DATA[SI][BX]
MOV AX, [BX + DATA][SI]
MOV AX, [BX + SI + DATA]
MOV AX, [BX]DATA[SI]
MOV AX, [BX + SI]DATA
```

同样地,基址-变址-相对寻址也不允许在指令中同时出现两个基址寄存器或两个变址寄存器。即下列指令也是非法的:

```
MOV AX, DATA[SI][DI]
MOV AX, [BX][BP]DATA
```

8. 隐含寻址

在有些指令的操作码中,不仅包含了操作的性质,还隐含了部分操作数的地址。如乘法指令 MUL,在这条指令中只需指明乘数的地址,而被乘数以及乘积的地址是隐含且固定的。这种将一个操作数隐含在指令码中的寻址方式就称为隐含寻址。

例 5-10 指令“MUL BL”的功能是把 AL 中的内容与 BL 中的内容相乘,乘积送到 AX 寄存器。即 $(AL) \times (BL) \rightarrow AX$ 。这条指令隐含了被乘数 AL 及存放结果的累加器 AX。

5.2.2 寻找转移地址的寻址方式

在一般情况下,当 BIU 取走一条指令后,指令指针 IP 会自动加 1 指向代码段中下一条要执行指令的地址,使程序按照预先设定好的次序,由低地址到高地址顺序执行。但有时需要改变程序的这种执行顺序,而转移到一个新的地址再顺序执行。因 BIU 是严格按照 CS 和 IP 所给出的地址去取指令的,所以,只要修改 CS 和 IP 的内容就能够实现程序转移到新地址继续执行的目的。寻找转移地址的寻址方法就是找出转移的目标地址,也可以说是下一条要执行指令的地址,而不再是操作数地址。

程序的转移可以是在当前代码段内,这种转移称为段内转移;程序也可以转移到另一个代码段,这称为段间转移。因此转移地址的寻址相应地有段内寻址和段间寻址两种方式。

1. 段内寻址方式

1) 段内直接寻址

段内直接寻址也称为相对寻址。转移的地址是当前 IP 的内容加上指令给定的 8 位或 16 位位移量,形成新的 IP,并使 CS 的内容保持不变,如图 5.9(a)所示。位移量可以为正,也可以为负。如果位移量是 8 位,称为段内直接短转移,转移范围为 $-128 \sim +127$;若位移量为 16 位,称为段内直接近转移,转移范围为 $-32\,768 \sim +32\,767$ 。

指令中常用字符标号指定位移量,即由汇编程序计算程序中的标号与转移指令之间的距离,也就是位移量。

段内直接寻址方式适合于条件转移或无条件转移类指令,但条件转移指令只能是段内短转移。

2) 段内间接寻址

此时程序转移的地址存放在寄存器或存储器的连续两单元中。执行时用寄存器或存储单元的内容取代当前 IP 的值,如图 5.9(b)所示。存储单元在数据段,其偏移地址根据转移指令指定的寻找操作数的寻址方式得出。

2. 段间寻址方式

使用段间寻址方式是表示程序转移的目标地址不在当前代码段内。亦即不仅要改变指令指针 IP 的内容,代码段寄存器 CS 的内容也要修改。段间寻址同样可分为直接寻址和间接寻址两种方式。

1) 段间直接寻址

这种寻址方式是直接用指令中给出的 16 位的段地址以及 16 位的偏移地址来取代当前 CS 和 IP 的内容,如图 5.9(c)所示。

2) 段间间接寻址

此时转移的目标地址存放在存储器的连续 4 个单元中,高地址单元存放新的 CS 值,低地址单元存放新的 IP 值。这 4 个单元在数据段中,它们的偏移地址也同样根据转移指令指定的寻址方式得出,如图 5.9(d)所示。

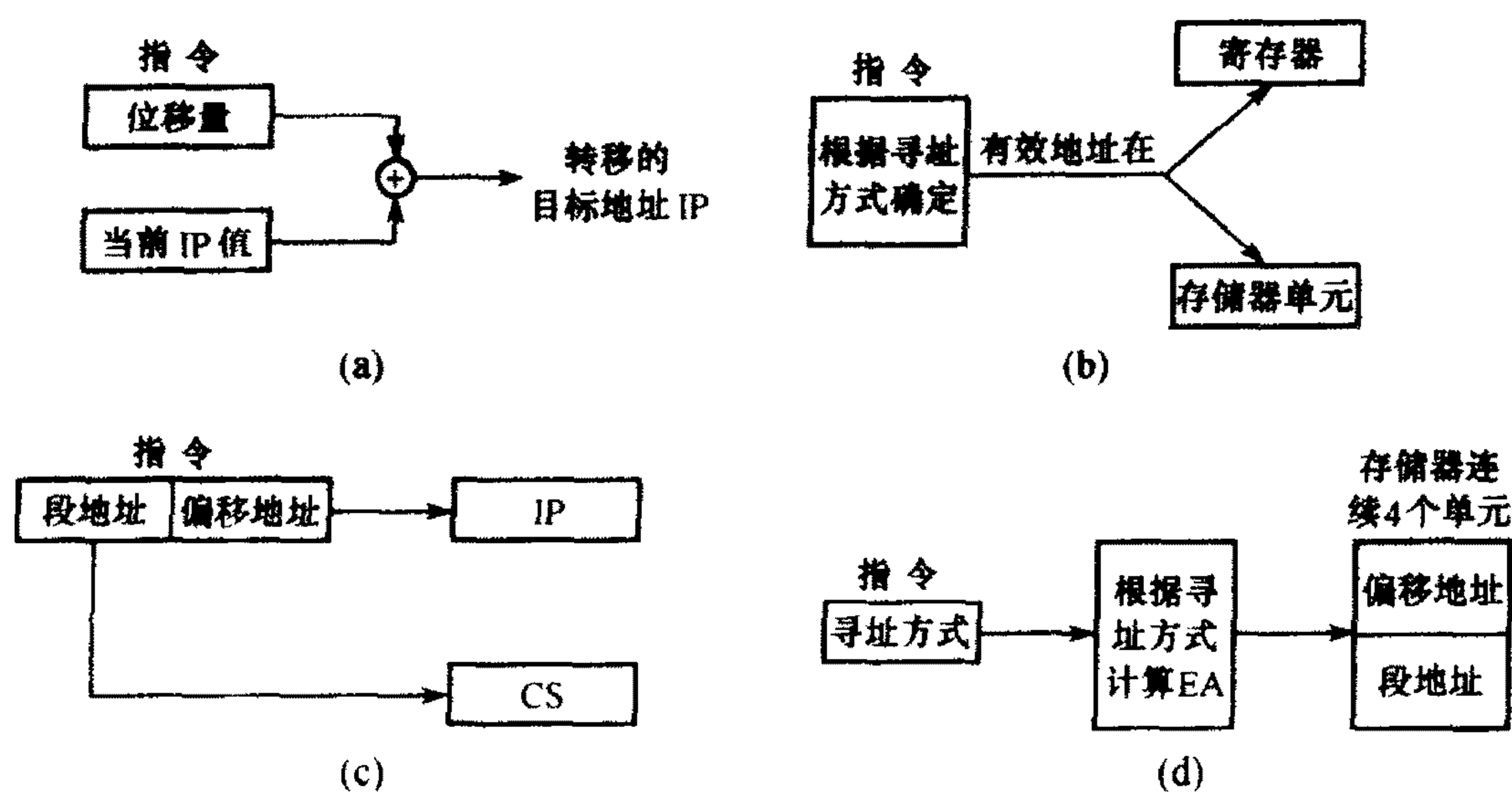


图 5.9 程序转移地址的寻址方式

5.3 8086 指令系统

一台计算机的指令系统通常有几十到几百条指令,它们决定了计算机的基本功能。8086/8088CPU 共有 96 条指令,其常用指令及助记符见表 5-5。按照它们的功能大致可分为以下六大类:

- ① 数据传送;
- ② 算术运算;
- ③ 逻辑运算和移位;
- ④ 串操作;
- ⑤ 程序控制;
- ⑥ 处理器控制。

下面分别说明各类指令的格式、执行的操作及其功能。

表 5-5 8086/8088 CPU 常用指令一览表

指令类型		助 记 符
数据 传 送	一般数据传送	MOV, PUSH, POP, XCHG, XLAT, CBW, CWD
	输入/输出指令	IN, OUT
	地址传送指令	LEA, LDS, LES
	标志传送指令	LAFH, SAFH, PUSHF, POPF
算 术 运 算	加法指令	ADD, ADC, INC
	减法指令	SUB, SBB, DEC, NEG, CMP
	乘法指令	MUL, IMUL
	除法指令	DIV, IDIV
	十进制调整指令	DAA, AAA, DAS, AAS, AAM, AAD
逻辑运算和移位指令		AND, OR, NOT, XOR, TEST, SHL, SAL, SHR, SAR, ROL, ROR, RCL, RCR
串操作		MOVS, CMPS, SCAS, LODS, STOS
控制转移指令		JMP, CALL, RET, LOOP, LOOPE, LOOPNE, INT, INTO, IRET, 各类条件转移指令
处理器控制指令		见表 5-7

5.3.1 数据传送指令

数据传送类指令是程序中使用最为频繁的一类指令,因为无论什么样的程序,都需要将

原始数据、中间运算结果、最终结果及其他信息在 CPU 的寄存器和存储器之间进行多次传送。数据传送时,数据从源地址传送到目标地址,且源地址中的数据保持不变,即相当于数据拷贝的操作。

数据传送指令一次可传送一个字节或一个字(对一批数据的传送由串操作指令完成),如果是交换指令,还可实现源操作数同目标操作数之间的互换。传送指令绝大多数都不会对状态寄存器 FLAGS 的标志位产生影响。

数据传送类指令包括:

- ① 通用数据传送;
- ② 堆栈操作;
- ③ 目标地址传送;
- ④ 输入/输出;
- ⑤ 标志传送。

1. 通用数据传送指令

通用数据传送指令包括一般传送指令 MOV、交换指令 XCHG、查表转换指令 XLAT 和字节扩展指令。

1) 一般传送指令 MOV

MOV 指令是双操作数指令,指令格式为

MOV dest,src

其中,dest 表示目标操作数,src 表示源操作数(以后同)。指令的功能是将一个操作数从源地址传送到目标地址,而源地址中的操作数保持不变。

MOV 指令是最普通、最常用的传送指令,它具有如下几个特点:

- ① 一次可实现一个字或一个字节数据的传送;
- ② 指令中的源操作数和目标操作数可以是通用寄存器、段寄存器、存储器单元及立即数,但不允许两个操作数同时为存储器操作数,也不允许当目标操作数是段寄存器时源操作数是立即数;
- ③ 可使用 5.2 中讨论过的各种寻找操作数的寻址方式。

例如:

;寄存器与寄存器或寄存器与段寄存器之间的传送

MOV BX,SI	;变址寄存器 SI 中的内容送到基址寄存器 BX
MOV DS,AX	;累加器 AX 的内容送到段寄存器 DS
MOV AX,CS	;段寄存器 CS 的内容送到累加器 AX
MOV AL,CL	;通用寄存器 CL 中的内容送 AL
MOV DL,CH	;通用寄存器 CH 中的内容送 DL

;寄存器与存储器之间的传送

MOV [BX], AX ;将 AX 的内容送存储单元,该单元在 DS 段,偏移地址 = (BX)
;若有 (DS) = 6000H, (AX) = 1234H, (BX) = 1200H, 则该条指令执行
;后, (61200H) = 34H, (61201H) = 12H。

MOV CL, [BP][DI] ;将 SS 段的偏移地址为 (BP) + (DI) 的存储单元的内容送 CL

MOV AX, [6000H] ;将 DS 段的 6000H 和 6001H 两个单元的内容送 AX

;立即数到寄存器的传送

MOV AL, 5 ;将立即数 5 送累加器 AL

MOV BX, 3078H ;将立即数 3078H 送寄存器 BX

;立即数到存储器的传送

MOV BYTE PTR [BP + SI], 5 ;将立即数 5 送 SS 段的偏移量为 (BP + SI) 的单元中

MOV WORD PTR [BX], 1005H ;将立即数 1005H 送 DS 段中偏移地址为 (BX) 和
;(BX + 1) 的两个存储单元

;存储器与段寄存器之间的传送

MOV DS, [1000H]

MOV [BX], ES

;若执行前 (DS) = 8000H, (ES) = 4000H, (BX) = 1200H, (81000H) = 00H, (81001H)
= 20H, 则以上两条指令执行后, (DS) = 2000H, (21200H) = 00H, (21201H) = 40H。

MOV 指令不能直接实现的传送有:

- ① 字到字节或字节到字的数据传送;
- ② 两个存储器单元之间的数据传送;
- ③ 两个段寄存器之间的数据传送;
- ④ 当源操作数是段寄存器时, 目标操作数不能是立即数;
- ⑤ 指令指针 IP 及代码段寄存器 CS 的内容不通过 MOV 指令修改;
- ⑥ 虽然许多指令的执行都对状态寄存器 FLAGS 的标志位产生影响, 但通常情况下,

FLAGS 整体不作为操作数。

2) 交换指令 XCHG

指令格式及操作:

XCHG OPRD1, OPRD2 ;(OPRD1) ↔ (OPRD2)

指令的操作是使源操作数与目标操作数的内容进行互换, 即将源操作数送到目标操作数, 同时将目标操作数传送到源操作数。

交换指令同样可以实现两地址中字节或字长操作数的互换, 但不能实现存储器单元之间及段寄存器之间内容的交换。如:

XCHG AX, BX ; (AX)→(BX), (BX)→(AX)

XCHG CL, DL ; (CL)→(DL), (DL)→(CL)

3) 查表转换指令 XLAT

XLAT 指令用于将内存某个连续区域中指定单元的内容送到累加器 AL。指令格式为

XLAT ; (AL)←((BX) + (AL))

或 XLAT src_table ; (src_table 表示要查找的表的首地址)

指令隐含的操作数 BX 的内容是指定区域的首地址(表首地址),使用时要将所查找单元的位移量(与表首间的距离)先送累加器 AL。

利用 XLAT 指令能够很方便地实现对一维数组表格的查找。

例 5-10 在内存的数据段中存放一张数值 0~9 的 ASCII 码转换表,首地址为 Hex_table,如图 5.10 所示。现要把数值 8 转换成对应的 ASCII 码,可用以下几条指令实现:

LEA BX, Hex_table ; (BX)←表首偏移地址

MOV AL, 8 ; (AL)←8

XLAT Hex_table ; 查表转换

结果 (AL) = 38H, 为 8 所对应的 ASCII 码。

使用查表转换指令时有一点要注意,由于要查找元素的序号放在 AL 中,所以表格的最大长度不能超过 256 B。

	:	
Hex_table+0	30	'0'
Hex_table+1	31	'1'
Hex_table+2	32	'2'
	:	
Hex_table+8	38	'8'
Hex_table+9	39	'9'
	:	

图 5.10 0~9 的换码表

4) 字位扩展指令

这是两条零操作数指令,用于带符号数字长的扩展,即在某些必要的场合,将符号数的符号位扩展到整个高 8 位或高 16 位。例如,要把有符号数 24H 扩展为一个字,则结果为 0024H;而如果要扩展的数是 84H,则结果为 FF84H。

扩展指令共有两条:

① 字节数扩展为双字节数的指令格式:

CBW

隐含的操作数为 AL 和 AH。当 (AL) < 80H 时, (AH) = 00H, 否则 (AH) = FFH。

② 字长数扩展为双字长数的指令格式:

CWD

指令中隐含了操作数 AX 和 DX, 扩展后的高 16 位放 DX 中。当 (AX) < 8000H 时, (DX) = 0000H, 否则 (DX) = FFFFH。

2. 堆栈操作指令

1) 堆栈的概念

堆栈(STACK)是由若干个连续存储单元组成的、操作时遵循先进后出原则的一个存储

器区,主要用于暂存中断和子程序调用时的现场数据及返回地址。它在内存中所处的段称为堆栈段,其段基地址放在堆栈段寄存器 SS 中。堆栈区中最高地址单元称为栈底,它是固定不变的,入栈的数据从高地址向低地址方向存放,存放的最低地址处称为栈顶,它的位置(即偏移地址)由堆栈指针 SP(Stack Pointer)表示,如图 5.11 所示。

堆栈操作具有如下特点:

① 操作只能在栈顶进行,每次入栈和出栈都必须是双字节数据(16 位)。即每进行一次入栈操作,SP 减 2;每进行一次出栈操作,SP 加 2;

② 操作遵循“先进后出(FILO)”的原则。最后压入堆栈的数据会最先被弹出。就相当于手枪的子弹匣,最后压入的子弹总是首先被射出。

将堆栈操作设计为“先进后出”是由堆栈的功能决定的。我们知道,在程序运行中,常碰到子程序的调用或响应中断而进入中断服务程序,此时就需要暂时终止程序的顺序执行,而转去执行子程序或中断服务程序,且在执行完后需能返回到原程序被中断处继续执行。为保证返回,就要求在转子程序前必须保存返回地址(也称断点),即调用子程序(或中断服务程序)的那条指令的下一条指令的 CS 和 IP 的内容。另外,要使返回后不影响原程序的正常执行,还要保存现场数据,即子程序中要用到的寄存器和存储器单元原来的值。

另外,在执行子程序(或中断服务程序)的过程中,有可能又会需要调用子程序(或中断服务程序),称为子程序的嵌套,如图 5.12 所示。这样就需要保护多个断点,并保证要能够逐个正确返回,于是就要求后保存的先取出,这就是为什么堆栈的操作是“先进后出”。

2) 堆栈操作指令

用于访问堆栈的指令共有两条,压入堆栈(压栈)指令 PUSH 和弹出堆栈(出栈)指令 POP。

(1) 指令格式:

压入堆栈指令格式:

PUSH OPRD ;将 OPRD 中的一个双字节数压入堆栈

弹出堆栈指令格式:

POP OPRD ;从堆栈中弹出一个双字节数送地址 OPRD 中

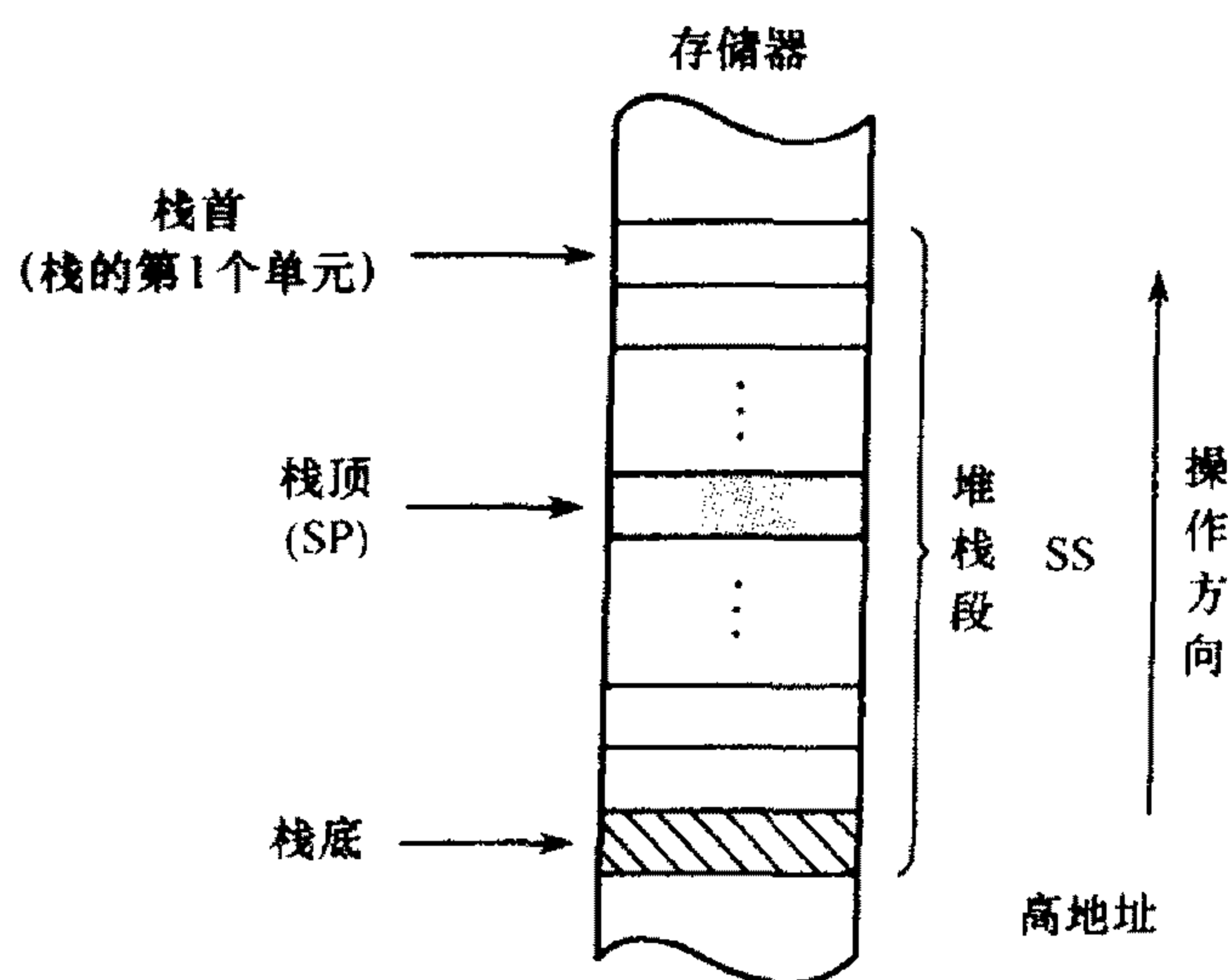


图 5.11 堆栈区域结构示意图

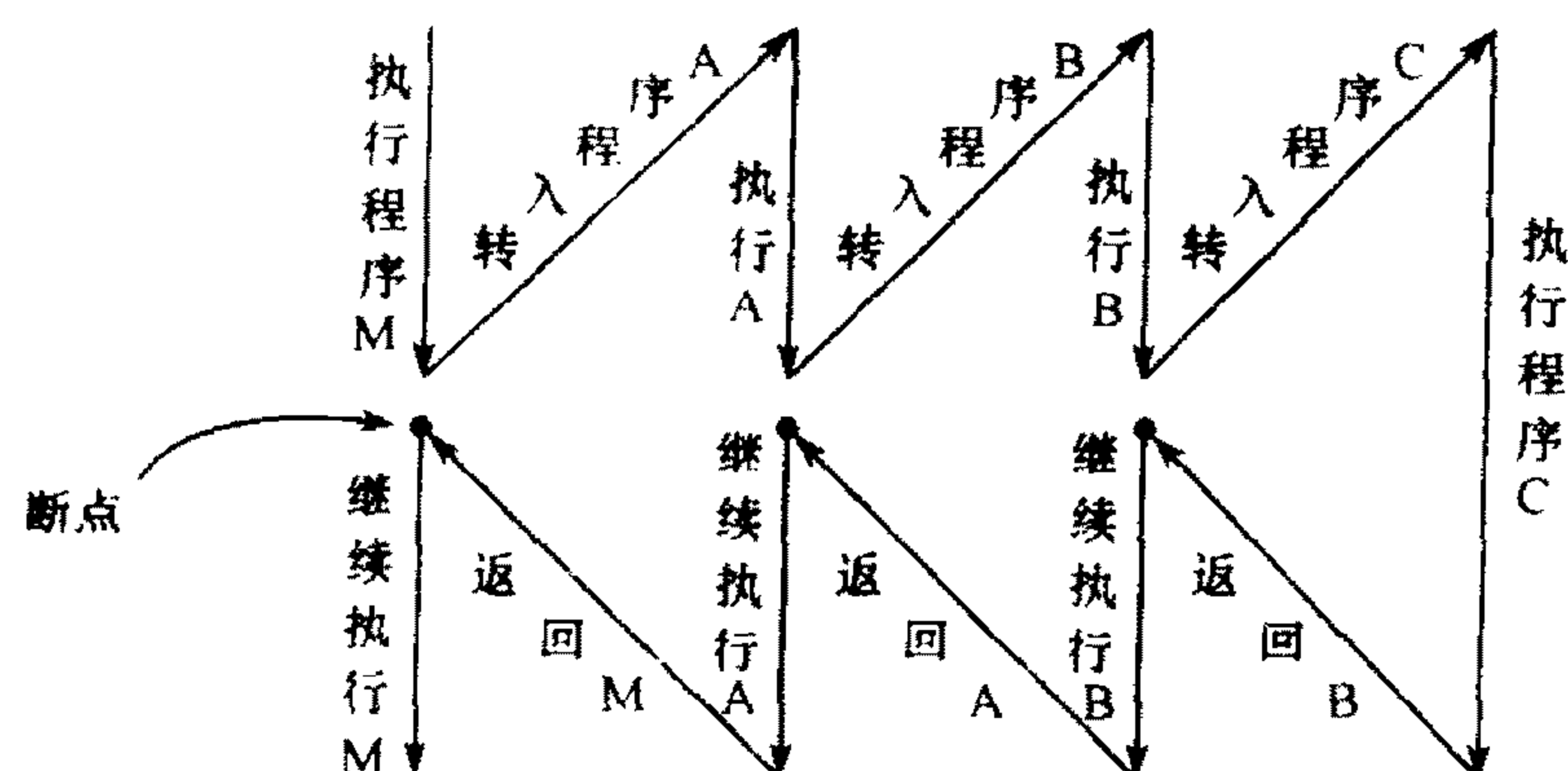


图 5.12 子程序调用示意图

操作数 OPRD 可以是：

- ① 通用寄存器。
- ② 段寄存器(CS 除外。PUSH CS 指令是合法的,而 POP CS 指令是非法的)。
- ③ 存储器单元。

操作数的类型必须是字操作数(16 位字长)——16 位寄存器或两个地址连续的存储单元。例如：

PUSH AX	;通用寄存器推入堆栈
PUSH BP	;基址指针寄存推入堆栈
PUSH DATA[SI]	;两个连续的存储单元推入堆栈
POP DS	;从堆栈弹出到段寄存器
POP [BX]	;从堆栈弹出到两个连续的存储单元

(2) 指令的执行过程

对压栈指令 PUSH,其执行过程是：

$(SP) - 2 \rightarrow (SP);$
 $OPRD \text{ 高 } 8 \text{ 位} \rightarrow [(SP) - 1];$
 $OPRD \text{ 低 } 8 \text{ 位} \rightarrow [(SP) - 2].$

即首先将堆栈指针 SP 的内容减 2,留出两个字节空间,再将地址 OPRD 中的双字节数送到 SP 指定的单元。图 5.13 所示为执行 PUSH AX 指令前后堆栈区的情况示意图。这里设 $(AX) = 1122H$ 。

由图可见,PUSH 指令是将一个字的源操作数送到堆栈的顶部。

出栈指令 POP 的执行过程为：

$[SP] \rightarrow OPRD \text{ 低 } 8 \text{ 位};$

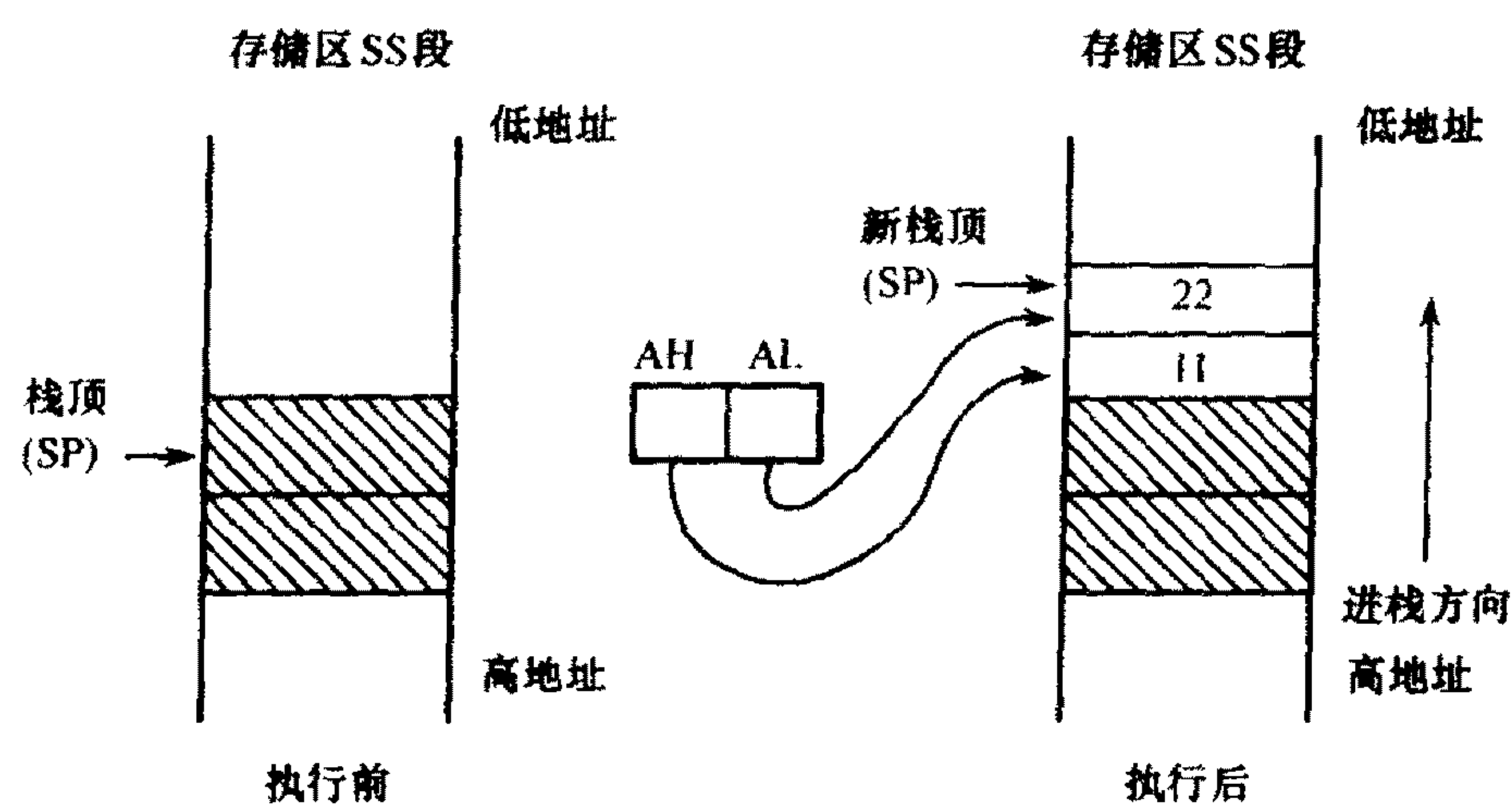


图 5.13 PUSH AX 指令执行示意图

$[SP + 1] \rightarrow \text{OPRD 高 8 位};$
 $(SP) + 2 \rightarrow (SP)。$

图 5.14 表示执行 POP AX 指令前后堆栈区的情况示意图。这里依然设 $(AX) = 1122H$ 。可见, POP 指令是将当前栈顶的一个字送到指定的目标地址中(如图中的 AX),并在执行后同时修改堆栈指针,以使 SP 始终指向新的栈顶位置。

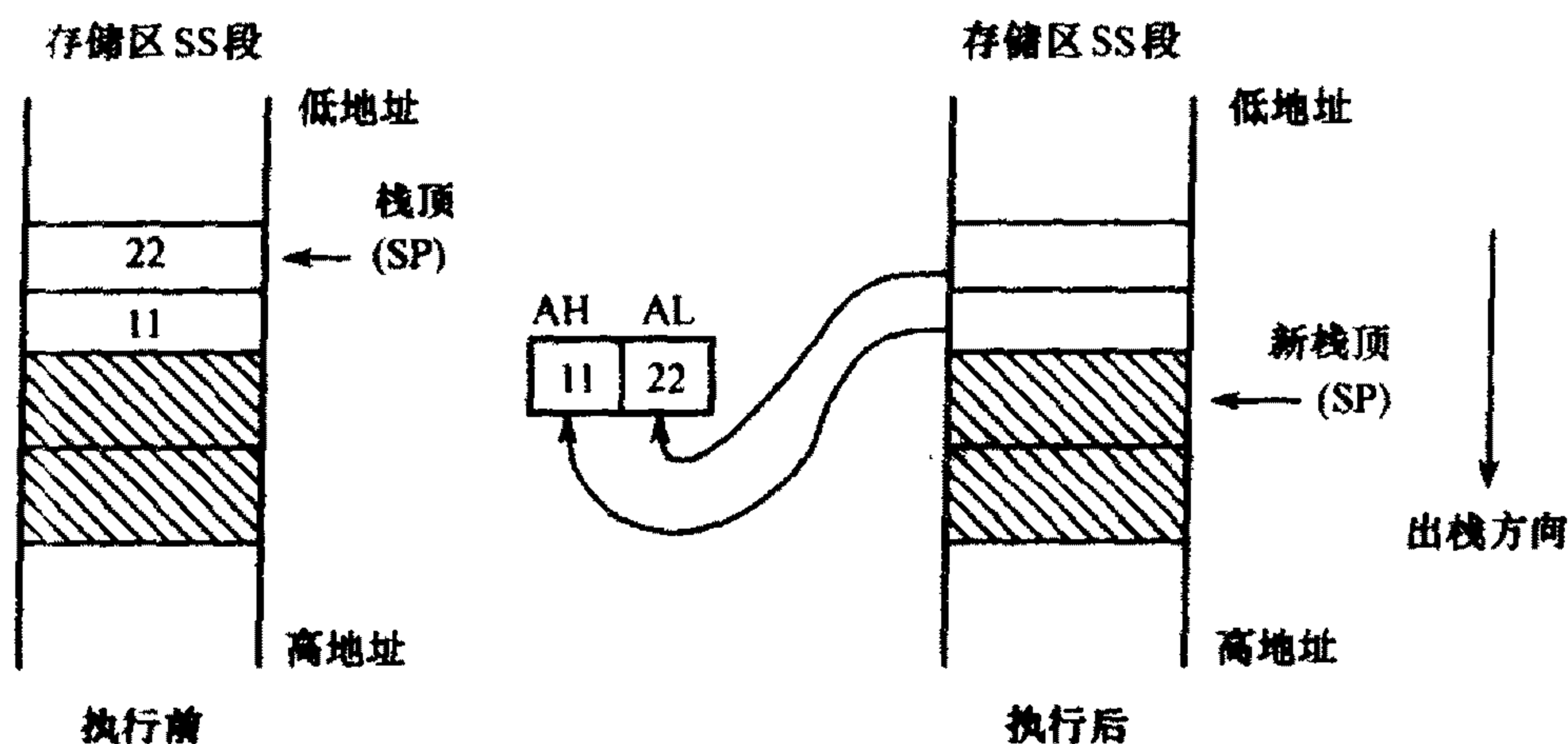


图 5.14 POP AX 指令执行示意图

在汇编语言程序中, PUSH 和 POP 指令一般成对出现,且执行顺序相反,以保持堆栈原有状态。当然,在必要时也可通过修改 SP 的值来恢复堆栈原有状态。

堆栈除在子程序调用和响应中断时用于保护断点地址、状态标志和现场信息外,还有一个重要的作用,就是用于子程序调用时的参数传递。在利用堆栈传递参数时,先将需要传递

的参数压入堆栈,然后调用子程序。为了在子程序中方便地访问到堆栈中的任一参数,可设置一个指向参数区的参数指针,利用相对寻址方式就可访问堆栈区域中的任何一个参数,而不再受“先进后出”原则的限制。有关这方面更进一步的内容请参考汇编语言程序设计方面的书籍。

3. 地址传送指令

地址传送指令的功能是将一个变量在内存中的地址送到指定的寄存器。偏移地址可送16位通用寄存器中,段地址则放于相应的段寄存器中。8086指令系统包括三条传送地址的指令:LEA、LDS和LES。

(1) LEA

指令格式:

LEA reg16, mem

LEA指令将变量mem的16位偏移地址送到指定的寄存器,该寄存器常用来作为地址指针。故这里的寄存器一般选用4个间址寄存器。例如:

LEA BX, BUFFER ;将内存单元 BUFFER 的偏移地址送 BX

MOV AL, [BX] ;取出 BUFFER 中的第1个数据送 AL

MOV AH, [BX + 1] ;取出 BUFFER 中的第2个数据送 AH

(2) LDS

指令格式:

LDS reg16, mem

LDS指令是将数据段中偏移地址为mem的连续4个单元的内容取出,低地址两单元中的数据送到BX、BP、SI或DI四个间址寄存器之一,高地址单元的数据送到数据段寄存器DS。

该指令用于确定一个32位的远地址指针(包括偏移地址和段地址),即要取的操作数在另一个数据段中。

例5-11 设(DS) = 6000H,内存地址为60348H开始的4个单元中存放了一个32位的远指针98011H,如图5.15所示。以下指令将该指针装入DS:SI中:

LDS SI, [0348H]

MOV AX, [SI]

指令执行后:

(SI) = 8011H, (DS) = 9000H, (AX) = 3412H。

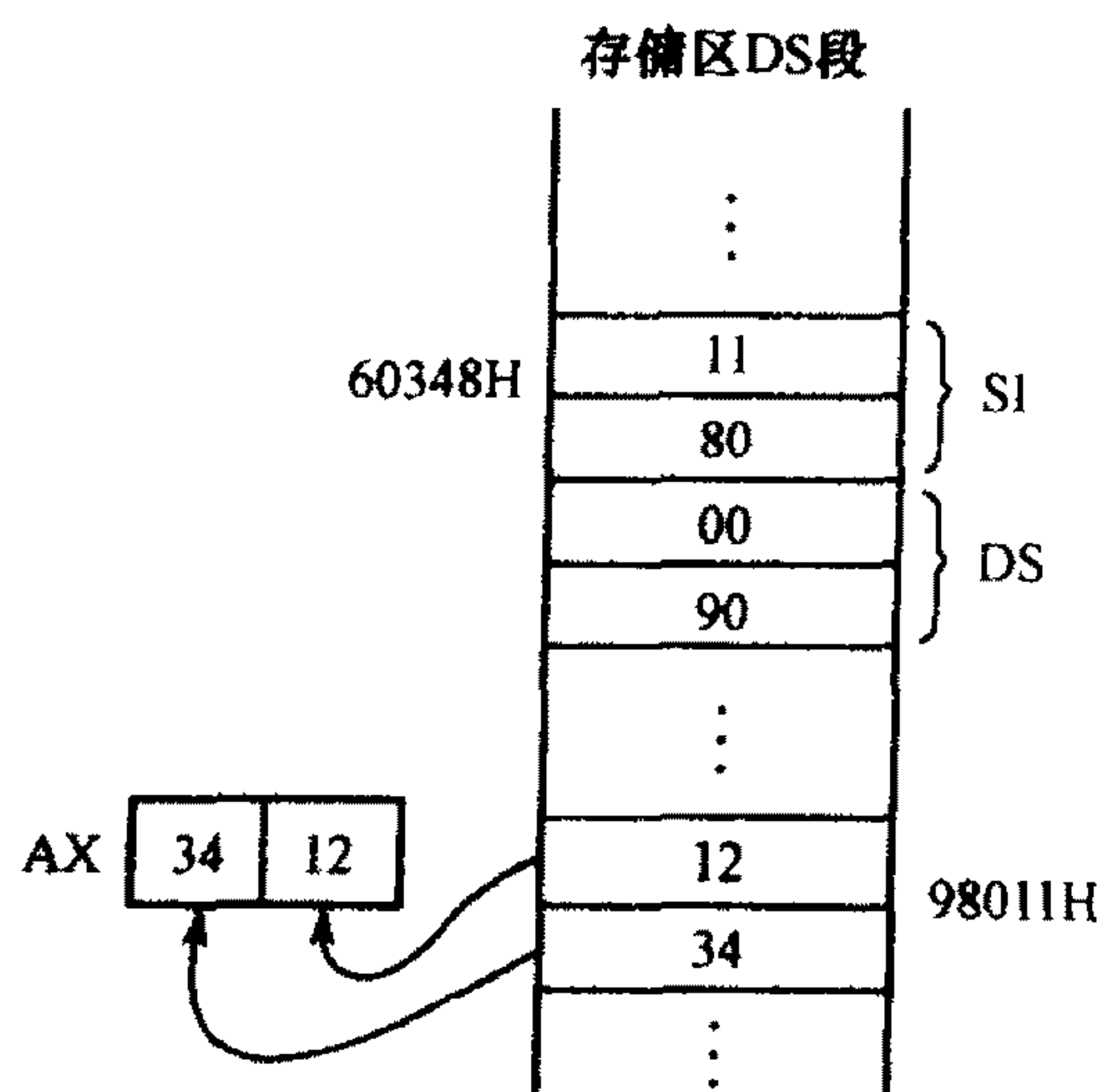


图 5.15 LDS 指令应用示意图

3) LES 指令

这条指令的格式及功能与 LDS 指令非常相似,不同的是,两个高地址单元中给出的段地址不是送到 DS,而是送到 ES。

4. 输入/输出(I/O)指令

计算机所处理的一切原始数据和所执行的程序,除固化在 ROM 中的之外,都是由输入设备送入的,而处理的结果则需要通过输出设备输出。因此,输入/输出指令的使用频率是非常高的,是指令系统中及其重要的一类指令。I/O 指令共有两条: IN 和 OUT。

输入指令 IN 用于从 I/O 端口读数据到累加器 AL(或 AX)中,而输出指令 OUT 用于把累加器 AL(或 AX)的内容写到 I/O 端口。即从 CPU 方面看,只有累加器 AL(或 AX)才能与 I/O 端口进行数据传送,所以这两条指令也称为累加器专用传送指令。

8086/8088 系统可连接多个外设端口,这些端口像存储器一样用不同的地址来区分。根据所寻址端口数量的不同,I/O 指令具有以下两种寻址方式:

① 直接寻址方式 指令中包含了一个 8 位的 I/O 端口地址,允许寻址 256 个端口,端口地址为 0 ~ FFH;

② 寄存器间接寻址方式 端口地址由 16 位寄存器 DX 指定,可寻址 64K 个端口(0 ~ FFFFH)。

1) 输入指令 IN

指令格式:

IN acc, port ;直接寻址, port 为 8 位立即数表示的端口地址

或

IN acc, DX ;间接寻址, 16 位端口地址由 DX 给出

指令从端口输入一个字节到 AL 或输入一个字到 AX 中。具体形式有以下四种:

IN AL, DATA ;端口地址 8 位, 输入一个字节

IN AX, DATA ;端口地址 8 位, 输入一个字

IN AL, DX ;端口地址 16 位, 输入一个字节

IN AX, DX ;端口地址 16 位, 输入一个字

例 5-12

MOV DX, 03B0H ;将 16 位端口地址送 DX

IN AL, DX ;从地址为 3B0H 的端口输入一个字节到 AL

2) 输出指令 OUT

指令格式:

OUT port, acc ;直接寻址, Port 为 8 位立即数表示的端口地址

或

OUT DX, acc ;间接寻址,16位端口地址由DX给出

指令把AL(或AX)的内容输出到指定的端口。输出指令的四种具体形式如下:

OUT DATA, AL ;端口地址8位,输出一个字节

OUT DATA, AX ;端口地址8位,输出一个字

OUT DX, AL ;端口地址16位,输出一个字节

OUT DX, AX ;端口地址16位,输出一个字

例5-13

OUT 43H, AL ;将AL的内容输出到地址为43H的端口

或

MOV DX, 43H ;端口地址43H送DX

OUT DX, AX ;将AX的内容输出到地址为43H的端口

注意,间接寻址的IN/OUT指令只能用DX寄存器作为间址寄存器。

4. 标志传送指令

这类指令是针对标志寄存器FLAGS操作的。在指令中FLAGS一般不作为操作数出现,它的各标志位的状态由程序运行结果决定。但在某些特殊情况下也会需要人工干预,即通过指令设置某些标志位的状态;另外,有时还需要将某些标志位的状态输出或保存。实现这些功能的指令有以下4条:

1) LAHF 指令

LAHF指令将标志寄存器FLAGS中的5个标志位——SF(符号标志)、ZF(零标志)、AF(半加进位标志)、PF(奇偶标志)和CF(进位标志)的状态分别传送到累加器AH的对应位,如图5.16所示。指令的执行对标志位没有影响。

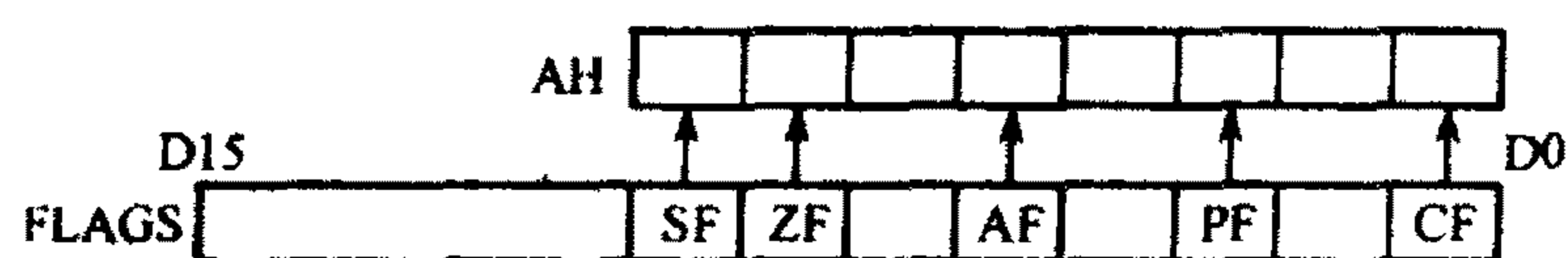


图5.16 LAHF指令操作示意图

2) SAHF 指令

SAHF指令与LAHF指令执行完全相反的操作,即将累加器AH中的第7、6、4、2、0位分别传送到标志寄存器FLAGS的对应位。如图5.17所示。该指令的执行显然会影响标志位SF、ZF、AF、PF和CF,它们将分别被AH的对应位的状态修改,但其他标志位不受影响。

下面的两条指令实际上属于堆栈操作指令,只不过它们的操作数不是别的,而是标志寄存器FLAGS。

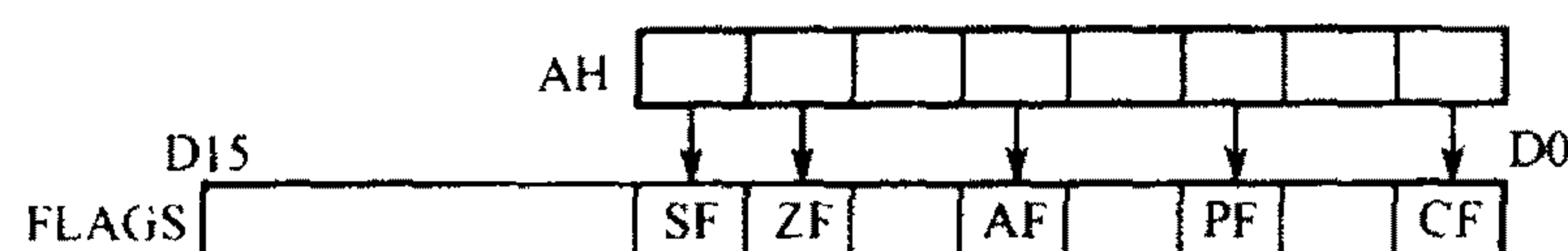


图 5.17 SAHF 指令操作示意图

3) PUSHF 指令

PUSHF 指令是将标志寄存器 FLAGS 压入堆栈。指令的操作为：

$$[SP - 1] \leftarrow (FLAGSH)$$

$$[SP - 2] \leftarrow (FLAGSL)$$

$$(SP) \leftarrow (SP) - 2$$

指令本身不影响标志位。

4) POPF 指令

POPF 指令的操作与 PUSHF 指令相反,它是将堆栈中当前栈顶的两个单元的内容弹出到标志寄存器 FLAGS。其操作为：

$$(FLAGSL) \leftarrow [SP]$$

$$(FLAGSH) \leftarrow [SP + 1]$$

$$(SP) \leftarrow (SP) + 2$$

POPF 指令影响标志位,它用栈顶两单元的内容替代了原标志寄存器的值。

PUSHF 指令和 POPF 指令都可用于在过程调用时保护标志位的状态,并在调用结束后恢复这些状态。PUSHF 和 POPF 指令一般是配对使用。

5.3.2 算术运算指令

这类指令也是一般计算机都具有的最基本、最常用的指令,主要针对的是定点数运算,运算的结果会对状态标志位产生影响。8086 提供了加、减、乘、除四组算术运算指令,可实现对字节或字、无符号数和有符号数的运算。指令有单操作数的,也有双操作数的。其中单操作数指令不允许使用立即数方式,即其操作数不能够是立即数;而双操作数指令中,立即数只能作为源操作数,这点与前面的数据传送类指令是相同的。此外,也不允许源操作数和目标操作数都是存储器。

算术运算可能涉及到运算结果是否溢出的问题。由第 2 章可知,无符号数同有符号数在表示方法、数的可表示范围及溢出标志等方面都不一样。有符号数的溢出是一种出错,而无符号数的溢出不能简单地认为是出错,也可看作是向更高位的进位。它们的判断标志分别为 CF 和 OF。

除这4组二进制的算术运算指令外,8086还提供了与之对应的4类十进制调整指令,可以将运算的结果(二进制)调整为用压缩BCD码或扩展BCD码表示的十进制数。

1. 加法运算指令

加法运算指令共有3条:不带进位位的加法指令ADD,带进位位的加法指令ADC及加1指令INC。所谓不带进位位或是带进位位,是指在运算时考虑或不考虑进位标志CF的状态。

1) 不带进位位的加法指令 ADD

ADD指令可实现两个字节数或两个双字节数相加,相加的两操作数可以是无符号数,也可以是带符号数。运算的结果送目标地址。

指令格式如下:

ADD OPRD1, OPRD2 ; (OPRD1) ← (OPRD1) + (OPRD2)

例如:

ADD CL, 20H ; (CL) ← (CL) + 20H

ADD AX, SI ; (AX) ← (AX) + (SI)

ADD DATA[BX], AL ; ((BX) + DATA) ← ((BX) + DATA) + (AL)

ADD DX, [BX + SI] ; (DX) ← (DX) + ((BX) + (SI))

加法指令的执行对全部6个状态标志位都会产生影响。

例 5-14

MOV AL, 7EH ; (AL) ← 7EH

ADD AL, 5BH ; (AL) ← 7EH + 5BH

这两条指令执行后,各状态标志位的状态分别为:AF = 1, CF = 0, OF = 1, PF = 0, SF = 1, ZF = 0。其中CF = 0表示最高位向前无进位,即若作为无符号数加法,其结果没有产生溢出;而OF = 1表示若作为有符号数加法,其运算结果产生溢出。事实上,指令执行后,(AL) = D9H > 7FH(8位带符号数的最大值),但D9H < FFH(8位无符号数的最大值)。

2) 带进位位的加法指令 ADC

ADC与ADD指令在功能、格式以及对标志位的影响上都基本相同,只是CF也要参加求和运算,即两操作数相加后还要再加上标志位CF的值。运算的结果依然送目标地址中。

指令格式如下:

ADC OPRD1, OPRD2 ; (OPRD1) ← (OPRD1) + (OPRD2) + (CF)

例 5-15 设CF = 1,写出以下指令执行后的结果。

MOV AL, 7EH

ADC AL, 0ABH

指令执行后:(AL) = 7EH + ABH + 1 = 2AH,且CF = 1。

ADC 指令主要用于多字节的加法运算。由于 8086 一次最多只能进行 16 位的加法运算,故对多于两个字节的数的加法,只能先加低 16 位(或低 8 位),再加高 16 位(或高 8 位),但在高位相加时,必须要考虑低位向上的进位,这时就要用到 ADC 指令。

3) 加 1 指令 INC

INC 指令用于将通用寄存器或存储器单元中的 8 位或 16 位数据加 1,结果依然送回该寄存器或存储单元。它的操作类似于 C 语言中的“++”运算符。

指令格式:

INC OPRD ;OPRD \leftarrow (OPRD) + 1

INC 指令不能对段寄存器内容或立即数进行修改。例如:

INC AX ;(AX) \leftarrow (AX) + 1

INC BL ;(BL) \leftarrow (BL) + 1

INC BYTE PTR[SI] ;将 SI 所指向的存储单元的内容 + 1,结果送回该单元

INC 指令不影响 CF 标志位,但对其他 5 个状态标志 AF、OF、PF、SF 及 ZF 会产生影响。它通常用于在循环程序中修改地址指针及循环次数等。

2. 减法指令

8086 共有 5 条减法指令,它们是:不考虑借位的减法指令 SUB,考虑借位的减法指令 SBB,减 1 指令 DEC,求补指令 NEG 以及比较指令 CMP。这 5 条指令的前 3 条分别对应于加法指令中的 ADD、ADC 和 INC 指令,它们的格式、运算的对象及运算结果对标志位的影响都与对应的加法指令相同,只是进行的是减法运算而已。

例如:

SUB BL,30 ;(BL) \leftarrow (BL) - 30H

SUB AL,[BP + SI] ;AL 的内容减去 SS 段(BP + SI)单元的内容,结果送 AL

SBB BL,30H ;(BL) \leftarrow (BL) - 30H - (CF)

SBB WORD PTR[SI],1034H ;((SI) + 1: (SI))单元的值减去立即数 1034H
;及(CF)值,结果送回((SI) + 1: (SI))单元

DEC AX ;(AX) \leftarrow (AX) - 1

DEC BL ;(BL) \leftarrow (BL) - 1

DEC BYTE PTR[DI] ;DI 所指单元内容减 1,结果送回该单元

与加法指令类似,SBB 指令也主要用于多字节的减法运算,而 DEC 指令则常用来在循环程序中修改循环次数。

以下两条指令是减法类指令所独有的:

1) 求补指令 NEG

指令格式:

NEG OPRD ; (OPRD) ← 0 - (OPRD)

NEG 指令的操作是用 0 减去操作数 OPRD, 结果送回该操作数。

指令把操作数 OPRD (可以是寄存器或存储器操作数) 视为补码表示的带符号数。之所以把 NEG 指令称为求补指令, 是因为对一个操作数取补码就相当于用 0 减去此操作数。

例如: 设 (DS) = 6000H, (BX) = 0010H, (60010H) = 47H。

NEG [BX]

执行上面的指令时, CPU 将进行如下减法运算:

$$\begin{array}{r} 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \\ -\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 1 \\ \hline 1\ 0\ 1\ 1\ 1\ 0\ 0\ 1 \end{array}$$

故指令执行后 (60010H) = B9H, 相当于对 47H 求补。换句话说, 利用 NEG 指令可以得到一个负数的绝对值。又如, 若 (AL) = FFH, 执行 NEG AL 后, 结果为 (AL) = 1 (FFH 是 -1 的补码)。

NEG 指令对 6 个状态标志位均有影响。要注意以下两点:

① 执行 NEG 指令后, 一般情况下都会使 CF 为 1。因为用零减去某个操作数, 自然会产生借位, 而减法的 CF 值正是反映无符号数运算中的借位情况。除非给定的操作数为零才会使 CF 为 0。

② 当指定的操作数的值为 80H (-128) 或为 8000H (-32768) 时, 执行 NEG 指令后, 结果不变, 即仍为 80H 或 8000H, 但 OF 置 1, 其他情况下 OF 均置 0。

2) 比较指令 CMP

CMP 指令用来比较两操作数的大小, 它进行的是减法操作, 但运算的结果并不送到任何提供地址, 即这条指令执行后两个操作数都保持不变, 而只影响标志位, 其后的程序正是根据标志位的状态决定下一步的走向。判断的方法为:

① 若两操作数是无符号数, 可根据 CF 的状态来确定。若 CF = 0, 则被减数大于减数 (被减数大则无需借位);

② 若两操作数是有符号数, 可通过 OF 和 SF 的状态确定。当 $OF \oplus SF = 0$ 时, 被减数大于减数; 当 $OF \oplus SF = 1$ 时, 减数大于被减数。

指令格式:

CMP OPRD1, OPRD2 ; (OPRD1) - (OPRD2), 结果不送回 OPRD1

例如:

CMP BX, 2100H ; (BX) - 2100H, 影响标志位

CMP CL, DH ; (CL) - (DH), 影响标志位

CMP AX, [BX + SI + 4] ; (AX) - ((BX + SI + 5) : (BX + SI + 4)), 影响标志位

在汇编语言程序中,比较指令的后面一般都紧跟一条条件转移指令,以根据比较结果决定程序的转向。

3. 乘法指令

乘法指令包括无符号数乘法指令 MUL 和有符号数乘法指令 IMUL 两条,可完成字节数与字节数相乘或双字节数与双字节数相乘,采用隐含寻址方式,隐含的目标操作数为 AX(与 DX),而源操作数由指令给出。

两个单字节数相乘,乘积为 16 位,存放于 AX 中;双字节数相乘,乘积为 32 位,高 16 位存放于 DX,低 16 位存放于 AX。

1) 无符号数的乘法指令 MUL

指令格式:

MUL OPRD

指令的操作为:

字节乘: $(AX) \leftarrow (OPRD) \times (AL)$

字乘: $(DX:AX) \leftarrow (OPRD) \times (AX)$

这里的源操作数 OPRD 可以是 8 位或 16 位的寄存器或存储器。如果乘积的高半部分(在字节相乘时为 AH,在字相乘时为 DX)不为零,则 $CF = OF = 1$,代表 AH 或 DX 中包含乘积的有效数字。否则 $CF = OF = 0$ 。其他标志位内容不定。例如:

MUL BX ; (DX:AX) \leftarrow (AX) \times (BX)

MUL BYTE PTR[SI] ; (AX) \leftarrow (AL) \times ((SI))

MUL DL ; (AX) \leftarrow (AL) \times (DL)

MUL WORD PTR[DI] ; (DX:AX) \leftarrow (AX) \times ((DI) + 1), (DI))

2) 带符号数的乘法指令 IMUL

IMUL 指令针对的是有符号数,在格式和功能上都与 MUL 指令类似。在运算时先将参加运算的数求变补(数为负则连符号位一起按位取反加 1,为正则不变),再将相乘后的结果求变补。若乘积的高半部分是低半部分的符号位的扩展,则 $CF = OF = 0$,否则 $CF = OF = 1$ 。

指令中的源操作数应满足带符号数的表示范围。

例 5-16 设 $(AL) = FEH$, $(CL) = 11H$,求 AL 与 CL 的乘积。

若将两个寄存器中内容看做无符号数,则应使用指令:

MUL CL

指令执行后: $(AX) = 10DEH$,因 AH 中的结果不为 0,故 $CF = OF = 1$ 。

若将两操作数看做有符号数,则二者相乘应采用有符号的乘法指令,即

IMUL CL

指令执行后: $(AX) = FFDEH = -34$ 。因 AH 中内容为 AL 中的符号扩展,故 $CF = OF = 0$ 。

4. 除法指令

与乘法指令相对应,除法指令也包括无符号数除法指令 DIV 和有符号数除法指令 IDIV,同样采用隐含寻址方式,隐含了被除数,而除数是由指令指定的寄存器或存储器单元。

与乘法指令不同的是,除法指令要求被除数的字长必须为除数字长的两倍。即如果除数是单字节数,则被除数就是累加器 AX 的内容;若除数为双字节数,则被除数必须是 32 位字长,高 16 位由 DX 给出,低 16 位由 AX 给出。除法指令对 6 个状态标志位均无影响。

1) 无符号数的除法指令 DIV

指令格式:

DIV OPRD

OPRD 是 8 位或 16 位的寄存器或存储单元的内容。指令的操作为:

字节除法:

$(AL) \leftarrow (AX) / (OPRD)$

$(AH) \leftarrow (AX) \% (OPRD)$ (%为取余数操作)

即 AX 中的 16 位无符号数除以 OPRD。得到的 8 位商放到 AL,8 位余数放到 AH。

字除法:

$(AX) \leftarrow (DX:AX) / (OPRD)$

$(DX) \leftarrow (DX:AX) \% (OPRD)$ (%为取余数操作)

即 DX:AX 中的 32 位无符号数除以 OPRD。得到的 16 位商放到 AX,16 位余数放到 DX 中。

若除法运算的结果大于寄存器可保存的值,即超出了 8 位或 16 位无符号数的可表示范围,则在 CPU 内部会产生一个类型 0 中断。另外,若被除数不够除数的双倍字长,可在其高位补 0。例如:

DIV BL ;(AX)除以(BL),商放(AL),余数放(AH)

DIV WORD PTR[SI] ;(DX):(AX)除以 SI 和 SI+1 所指向单元的内容
;商放(AX),余数放(DX)

2) 有符号数的除法指令 IDIV

当带符号数相除时,必须用 IDIV 指令。该指令在格式和功能上都和 DIV 指令类似。例如:

IDIV CX ;DX 和 AX 中的 32 位数除以(CX),商在 AX 中,余数在 DX 中

IDIV BYTE PTR[BX] ;(AX)除以 BX 所指单元中的内容,商在 AL 中,余数在 AH 中

IDIV 指令的结果,商和余数均为带符号数,且余数符号与被除数符号相同。如 -26 除以 +4,可得到两种结果,一种结果是商 -6,余数 -2;另一种结果是商 -7,余数 +2,这两种结果都是正确的,但按照 8086 指令系统的规定,会得到前一种结果。

如果被除数字长与除数字长相同,同样需要将其扩展。与无符号数的扩展方法不同,带符号数的扩展是将符号位扩展。可用下面介绍的字节扩展指令来实现。

5. BCD 码(十进制)运算调整指令

除以上 4 组基本算术运算指令外,8086 还提供了 6 条用于 BCD 码运算的调整指令。它们均采用隐含寻址方式,隐含的操作数是累加器 AL(或 AL 和 AH)。它们一般不单独使用,而是与加、减、乘、除指令配合使用,将运算的结果调整为用 BCD 码表示的十进制数。这 6 条指令分别为:

① DAA 压缩 BCD 码加法的十进制调整指令。用于对两个压缩 BCD 码相加之后的和(相加结果须放 AL 中)进行调整,以产生正确的压缩 BCD 码(因为按二进制的运算规则做十进制加法,当结果的 BCD 数的任意一位大于 9 或产生进位时,结果肯定不正确。对 BCD 数的减、乘、除也同样有此问题)。该指令常跟在 ADD 或 ADC 指令之后使用。

DAA 指令影响除 OF 外的其余 5 个状态标志位。

② AAA 非压缩 BCD 码加法的十进制调整指令。用于对两个非压缩(扩展)BCD 数相加之后的和(相加结果须放 AL 中)进行调整,形成一个正确的扩展 BCD 码,调整后的结果其低位在 AL 中,高位在 AH 中。

例 5-17 用 BCD 码计算 $9 + 4 = ?$ 。

```
MOV AL,09H      ;BCD 数 9
MOV BL,04H      ;BCD 数 4
ADD AL,BL       ;(AL) = 09H + 04H = 0DH
AAA             ;(AL) = 0DH + 06H = 03H(高 4 位清零);(AH) = 1;(CF) = 1
```

AAA 指令只影响 AF 和 CF 标志。

③ DAS 压缩 BCD 码减法的十进制调整。用于对两个压缩 BCD 码相减后的结果(在 AL 中)进行调整,产生正确的压缩 BCD 码。对标志位的影响与 DAA 指令相同。

④ AAS 非压缩 BCD 码减法的十进制调整。用于对两个非压缩 BCD 码数相减之后的结果(在 AL 中)进行调整,形成一个正确的非压缩 BCD 码,其低位在 AL 中,高位在 AH 中。

DAS 和 AAS 指令必须紧跟在减法指令 SUB 或 SBB 后使用,且 SUB 和 SBB 指令的执行结果必须在累加器 AL 中。两条指令对标志位的影响与 DAA、AAA 指令相同。

⑤ AAM 非压缩 BCD 码乘法的十进制调整指令。对两个非压缩 BCD 码数相乘的结果(AX 中)进行调整,以得到正确的结果。其操作的实质是把 AL 中的二进制数转换为十进制数,对于十进制值不超过 99 的二进制数,可用一条 AAM 指令实现二—十进制转换。

执行 AAM 指令前须先有一条 MUL 指令(BCD 码总视为无符号数)将两个非压缩 BCD 码相乘,结果放在 AX 中,然后用 AAM 指令进行调整,于是 AX 中就可得到正确的非压缩 BCD 码乘积。积的高位在 AH 中,低位在 AL 中。

例 5-18 用非压缩 BCD 码计算 $7 \times 9 = ?$ 。

```
MOV AL,07H      ;(AL) = 07H,即非压缩 BCD 数 7
MOV BL,09H      ;(AL) = 09H,即非压缩 BCD 数 9
MUL BL          ;(AX) = 07H × 09H = 003FH
AAM             ;(AX) = 0603H,即非压缩 BCD 数 63,(SF) = 0,(ZF) = 0,(PF) = 1
```

AAM 指令影响 PF、SF 和 ZF 标志位。

⑥ AAD 非压缩 BCD 码除法的十进制调整指令。以上 5 条十进制调整指令都是跟在相应的指令后面执行,而 AAD 指令则是在进行除法之前执行。即在两个非压缩 BCD 码相除之前,先用一条 AAD 指令进行调整,然后再用 DIV 指令。

例 5-19 计算 $23 \div 4 = ?$ 。

```
MOV AX,0203H    ;(AX) = 0203H,即非压缩 BCD 数 23
MOV BL,4         ;(BL) = 04H,即非压缩 BCD 数 4
AAD              ;(AX) = 02H × 0AH + 03H = 0017H
DIV BL           ;(AH) = 03H,(AL) = 05H,即结果为商 5 余 3
```

执行 AAD 后,(AH) = 0,(AL) = 17;再执行 DIV 指令后,(AH) = 03H,(AL) = 05H。

AAD 指令影响 PF、SF 和 ZF 标志位。

5.3.3 逻辑运算和移位指令

这一类指令包括逻辑运算指令和移位指令两大部分,移位指令中又分为非循环移位指令和循环移位指令。

1. 逻辑运算指令

8086 提供的逻辑运算指令共有 5 条,包括: AND(逻辑与)、OR(逻辑或)、NOT(逻辑非)、XOR(逻辑异或)及 TEST(测试)指令。这些指令可对 8 位或 16 位寄存器或存储器单元中的内容进行按位操作。除 NOT 指令外,其他 4 条指令对标志位的影响相同。它们的执行都会使 $CF = OF = 0$,AF 值不定,并对 SF、PF 和 ZF 有影响。NOT 指令对所有标志位都不影响。

1) AND 指令

这是一条双操作数指令,实现两操作数按位相与,并将与运算的结果送到目标地址中。两操作数的性质与 MOV 指令相同。指令格式:

```
AND OPRD1,OPRD2      ;(OPRD1) ← (OPRD1) ∧ (OPRD2)
```

例如:

```
AND AL,0FH           ;AL 中的内容与 0FH 相与,结果在 AL 中
AND CX,0FF00H        ;CX 中的内容与 FF00H 相与,结果在 CX 中
```


AND AX, BX ; AX 和 BX 中内容相与, 结果在 AX 中
 AND AX, [BX]DATA ; AX 中内容和地址为 (BX) + (DATA), (BX) + (DATA) + 1
 ; 两单元内容相与, 结果在 AX 中

AND 指令的主要用途是使目标操作数中的某些位保持不变的情况下, 把其他位清零。这需要指定一个屏蔽字, 屏蔽字的各位这样设置: 目标操作数中哪些位要清 0, 就把屏蔽字相对应的位设为 0, 其他位设为 1。如上例中第 1 行 AND AL, 0FH, 0FH 即是屏蔽字, 其高 4 位为 0, 低 4 位为 1, 表示将 AL 中的高 4 位清除, 而低 4 位保留。

如果使某一操作数自身相与, 操作数本身并不改变, 但会使标志位 CF = OF = 0。例如:

AND AX, AX

该指令执行后, AX 内容不变, 但使标志位 CF 和 OF 清零。

2) OR 指令

OR 指令的格式、操作数的性质及对标志位的影响都与 AND 指令相同, 只是实现的是两操作数按位相或的运算, 运算的结果同样送目标地址中。指令格式:

OR OPRD1, OPRD2 ; (OPRD1) ← (OPRD1) V (OPRD2)

例如:

OR AL, 20H ; AL 的内容和 20H 相或, 结果在 AL 中
 OR AX, 00FFH ; AX 中内容和 00FFH 相或, 结果在 AX 中
 OR [BX], AL ; (BX) 单元的内容和 AL 的内容相或, 结果送回 (BX) 单元

OR 指令的主要用途是使目标操作数的某些位保持不变, 而将另外一些位置 1。这时, 源操作数应这样设置: 目标操作数哪些位需要置为“1”, 就把源操作数中与之对应的位设为 1, 其他位则设为 0。

3) 求反指令 NOT

这是一条单操作数指令, 其执行的结果是使指定的操作数按位取反后再保存在原地址。指令格式:

NOT OPRD

这里, OPRD 可以是 8 位或 16 位的寄存器或存储器操作数, 但不能是立即数。NOT 指令对标志位不产生影响。例如:

NOT AX ; 将 AX 中内容按位取反, 结果送回 AX
 NOT CL ; 将 CL 中内容按位取反, 结果送回 CL
 NOT WORD PTR [SI] ; 将 (SI) 和 (SI + 1) 两单元中的内容按位取反, 再送回这两单元

4) 异或指令 XOR

指令格式:

XOR OPRD1, OPRD2 ; (OPRD1) ← (OPRD1) (OPRD2)

XOR 指令将源操作数和目标操作数按位进行异或运算,结果送回目标地址。例如:

XOR AX,1122H ;AX 的内容与 1122H 异或,结果在 AX 中

XOR AL,[BX] ;AL 的内容与(BX)单元的内容异或,结果送 AL

根据异或运算的性质,某一操作数和自身相异或,结果为零。在程序中常利用这一特性,使某寄存器清零。

5) 测试指令 TEST

TEST 指令的格式、操作数性质及完成的操作和 AND 指令基本类似,区别是与运算的结果不送回目标地址,而只是影响标志位。故这条指令常用于在不破坏原操作数内容的情况下检测操作数中某些位是“1”还是“0”。例如:

TEST AL,02H ;若 AL 中 D1 为 1,则 ZF=0,否则 ZF=1

TEST AX,8000H ;若 AX 中最高位为 1,则 ZF=0,否则 ZF=1

2. 移位指令

移位指令从功能上可分为非循环移位指令和循环移位指令两大类,其中非循环移位指令又包括算术移位和逻辑移位。另外从移动的方向上,又分为左移和右移两种。

1) 非循环移位指令

非循环移位指令共有 4 条:

- ① 算术左移指令 SAL(Shift Arithmetic Left);
- ② 算术右移指令 SAR(Shift Arithmetic Right);
- ③ 逻辑左移指令 SHL(Shift Logic Left);
- ④ 逻辑右移指令 SHR(Shift Logic Right)。

算术移位和逻辑移位的区别在于它们操作的对象不同,前者的操作数为带符号数,后者的操作数为无符号数。因为如此,在右移的操作上有所不同,算术右移指令始终保持最高位(即符号位)不变,而逻辑右移指令是在最高位补 0。

以上 4 条指令均可实现对寄存器或存储器操作数进行指定次数的移位,可以是字操作,也可以是字节操作。在移动 2 位或更多位时,移动的次数要由 CL 寄存器指定。

指令格式分别为:

SHL OPRD,1 ;逻辑左移指令

SHL OPRD,CL

SAL OPRD,1 ;算术左移指令

SAL OPRD,CL

SHR OPRD,1 ;逻辑右移指令

SHR OPRD,CL

SAR OPRD,1 ;算术右移指令

SAR OPRD, CL

SHL/SAL 指令执行的操作是将目的操作数的内容左移 1 位或 CL 所指定的位, 每左移 1 位, 左边的最高位移入标志位 CF, 而在右边的最低位补 0。

在移动次数为 1 的情况下, 若移位之后, 操作数的最高位与 CF 标志位的值不相等, 则溢出标志位 $OF = 1$, 否则 $OF = 0$ 。 $OF = 1$, 对 SHL 指令不表示左移后溢出, 而对 SAL 指令则表示移位后超出了符号数的表示范围。另外, 指令还影响标志位 PF、SF 和 ZF。

SHR 和 SAR 指令都是将操作数 OPRD 顺序向右移 1 位或 CL 指定的位数, 每右移一位, 右边的最低位移入标志位 CF, 但 SHR 指令针对的是无符号数, 移动后在左边的最高位补 0; 而 SAR 指令是将操作数视为有符号数, 右移后最高位不是补 0, 而是保持不变。

SHR 指令影响标志位 CF 和 OF。如果移动次数为 1, 且移位之后新的最高位和次高位不相等, 则标志位 $OF = 1$, 否则 $OF = 0$ 。若移位次数不为 1, 则 OF 状态不定。SAR 指令对标志位 CF、OF、PF、SF 及 ZF 均有影响, 但 AF 值不定。

以上 4 条指令的操作示意图如图 5.18 所示。

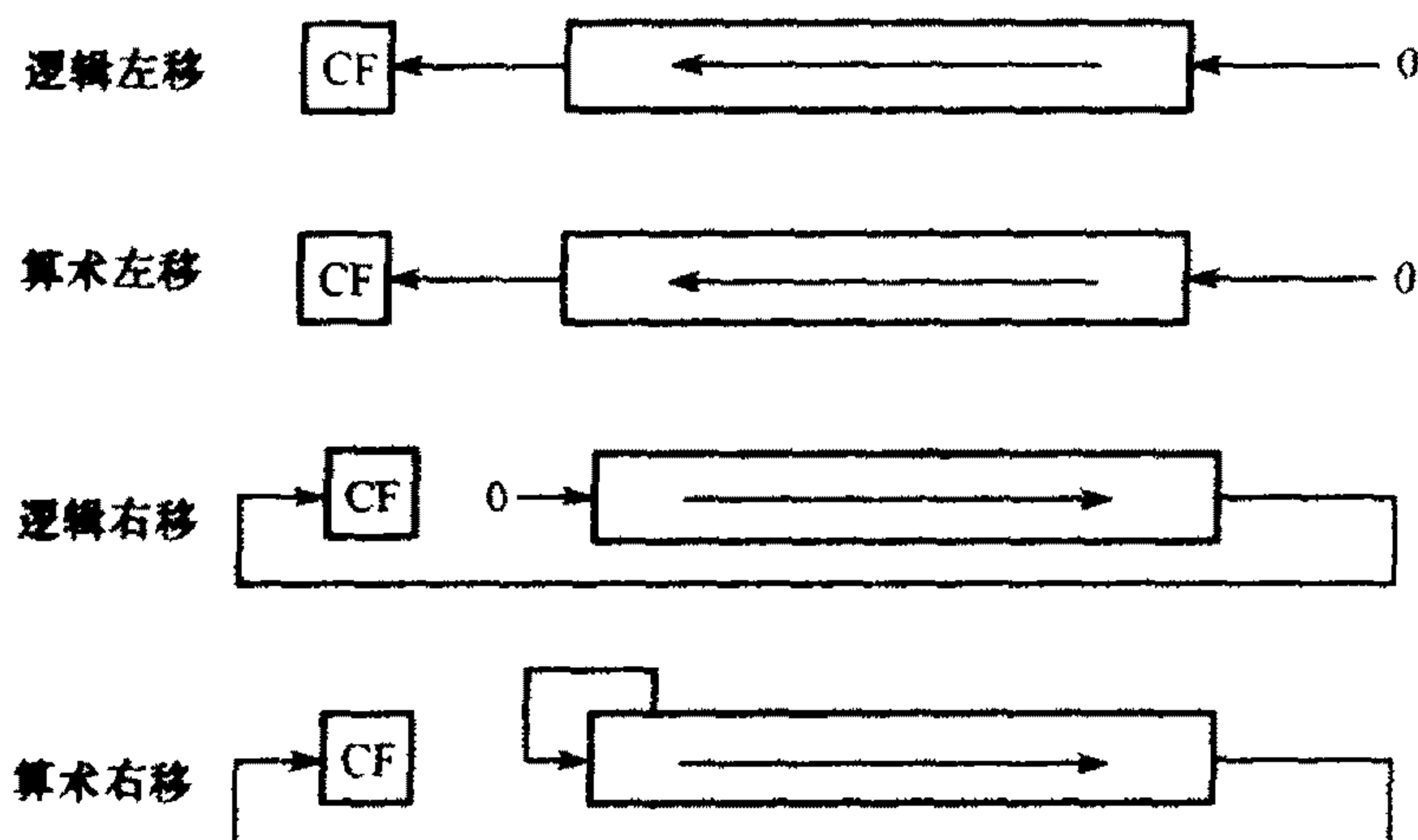


图 5.18 非循环移位操作示意图

非循环移位指令的一个重要作用, 是用于实现简单的乘除运算。算术左移或右移 n 位, 表示对带符号数乘以 2^n 或除以 2^n ; 同理, 逻辑左移或右移 n 位, 实现对无符号数乘以 2^n 或除以 2^n 的运算。移位指令的这个性质非常重要, 因为移位指令的执行时间要比乘除运算的短, 这样, 用移位指令来实现简单的乘除运算就会使运算速度提高很多。

例 5-20

MOV AL, 41H

SHL AL, 1

执行结果: $(AL) = 82H$, $CF = 0$, $OF = 1$ 。若视 82H 为无符号数, 则它没有溢出 ($82H <$

FFH);若视它为有符号数,则溢出(82H>7FH),因为移位后正数变成了负数。

例 5-21 用移位指令完成将一个双字节无符号数乘以 10 的运算。

因为 $10x = 8x + 2x = 2^3x + 2^1x$, 因此有如下程序:

```

LEA SI, DATA      ;存放数据单元的偏移地址送 SI
MOV AX, [SI]        ;(AX)←被乘数
SHL AX, 1           ;(AX) = DATA * 2
MOV BX, AX          ;暂存 BX
MOV CL, 3           ;(CL)←移位次数
SHL AX, CL          ;(AX) = DATA * 8
ADD AX, BX          ;(AX) = DATA * 10
HLT

```

2) 循环移位指令

循环移位指令按是否与进位标志位 CF 一起循环,还可分为不带进位位的循环移位及带进位位的循环移位两种,指令的操作数类型及格式与非循环移位指令完全相同,共有 4 条:

- ① 不带进位标志位 CF 的循环左移指令(格式: ROL OPRD, 1 或: ROL OPRD, CL);
- ② 不带进位标志位 CF 的循环右移指令(格式: ROR OPRD, 1 或: ROR OPRD, CL);
- ③ 带进位标志位 CF 的循环左移指令(格式: RCL OPRD, 1 或: RCL OPRD, CL);
- ④ 带进位标志位 CF 的循环右移指令(格式: RCR OPRD, 1 或: RCR OPRD, CL)。

循环移位指令一般用于实现循环式控制、高低字节互换或是与非循环移位指令一起实现双倍字长或多倍字长的移位。指令的操作示意如图 5.19 所示。

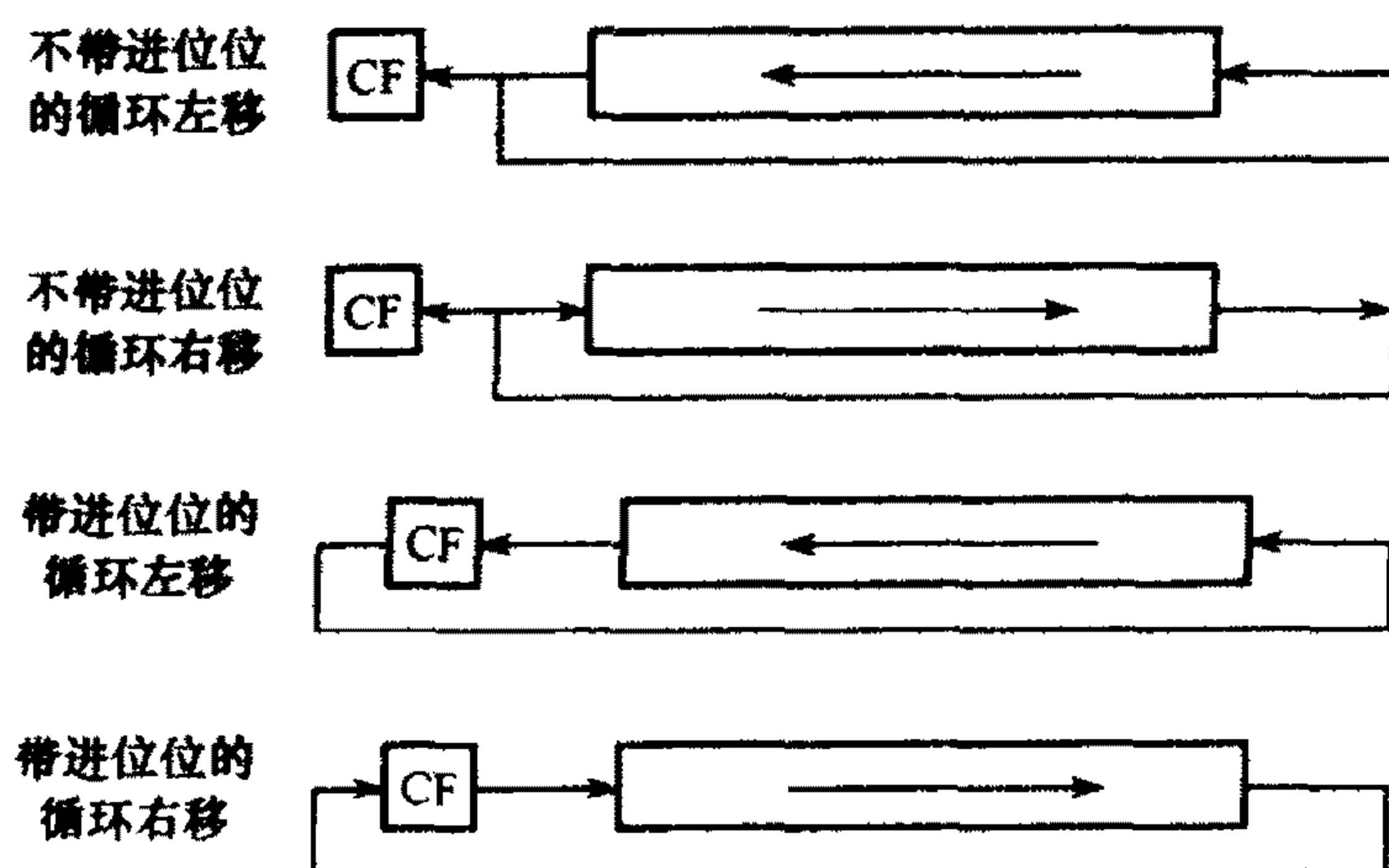


图 5.19 循环移位指令操作示意图

ROL 指令将目标操作数向左循环移动 1 位或由 CL 指定的位数,最高位移入 CF,同时再移入最低位构成循环,进位标志 CF 不在循环之内。指令影响标志位 CF 和 OF。如果循环移位次数为 1,且移位之后目标操作数的最高位和 CF 值不相等,则标志位 OF = 1,否则 OF = 0。若移位次数不为 1,则 OF 状态不定。

ROR 指令将目标操作数向右循环移动一位或 CL 指定的位数,最低位移入 CF,同时再移入最高位构成循环。指令影响标志位 CF 和 OF。如果循环移位次数为 1,且移位之后新的最高位和次高位不等,则标志位 OF = 1,否则 OF = 0。若移位次数不为 1,则 OF 状态不定。

RCL 指令将目标操作数连同进位标志位 CF 一起向左循环移动 1 位或 CL 指定的位数,最高位移入 CF,而 CF 原来的值移入最低位。指令对标志位的影响与 ROL 指令相同。

RCR 指令将目标操作数连同进位标志位 CF 一起向右循环移动 1 位或 CL 指定的位数,最低位移入 CF,而 CF 原来的值移入最高位。其对标志位的影响与 ROR 指令相同。

循环移位指令与非循环移位指令不同,循环移位后,操作数中原来各位数的信息不会丢失,而只是改变了位置而已(仍在操作数中的其他位置上或 CF 中),如果需要还可恢复(反向移动即可)。

例 5-22 指令: RCL BYTE PTR[100AH],1。

设(DS) = 6000H,且指令执行前(6100AH) = 8EH,CF = 0。

执行上面的指令后: (6100AH) = 1CH,CF = 1,操作示意如图 5.20 所示。

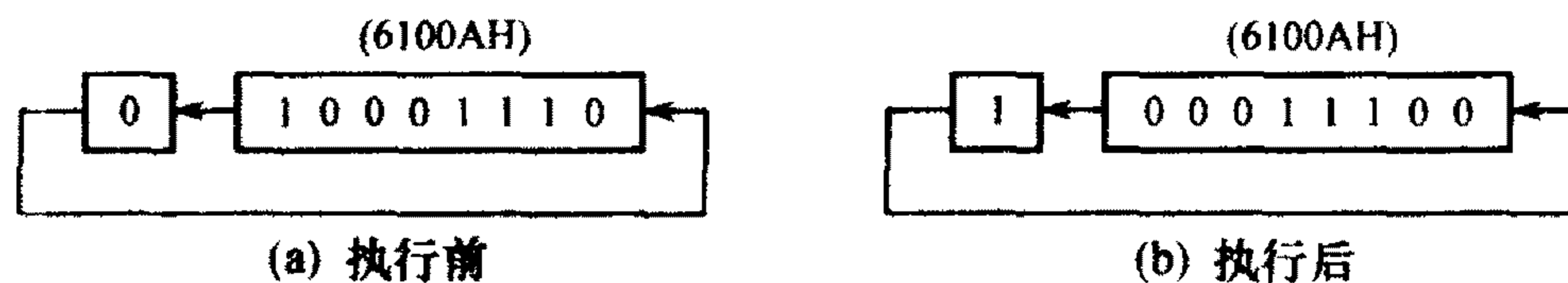


图 5.20 RCL 指令执行举例

例 5-23 将 DX 和 AX 两个寄存器组合成一个整体,使这 32 位操作数一起逻辑左移 1 位。即 AX 的最高位应移入 DX 的最低位,如图 5.21 所示。可用两条指令实现这个操作:

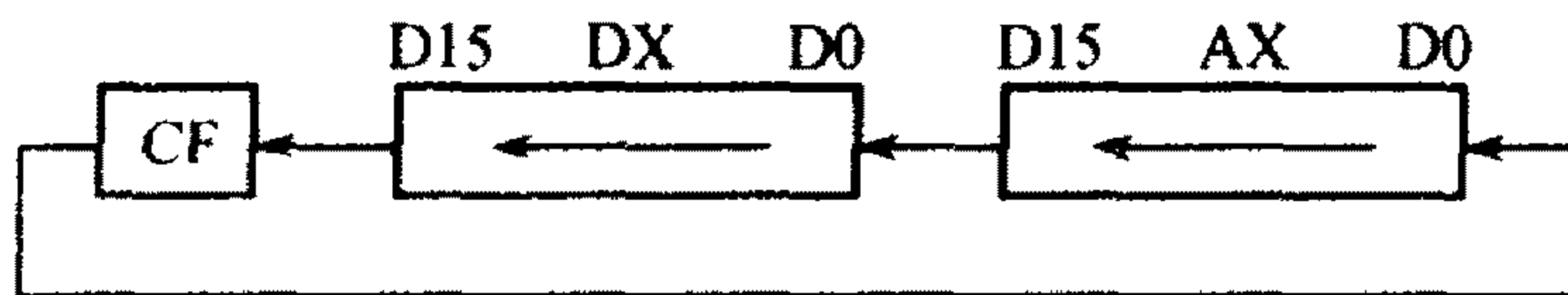


图 5.21 31 位寄存器左移 1 位

SHL AX,1 ; AX 左移 1 位,CF←DX 最高位

RCL DX,1 ;DX 带进位位循环左移 1 位,AX 最低位 \leftarrow CF

5.3.4 串操作指令

1. 指令特点

当计算机随着其技术的不断发展而逐渐进入各个应用领域时,人们对它的要求就不再限于只做科学计算,而希望能更多地应用于信息管理、数据处理、办公自动化等,即要求具有较强的非数值处理能力。串操作指令就是一种非数值处理指令,能够由硬件直接支持,用于数据块或字符串的处理。

所谓“串”,是指存储器中地址连续的若干单元字符或数据。串操作指令就是用来对串中每个字符或数据做同样操作的指令。8086CPU 的串操作指令包括“串传送”、“串比较”、“串扫描”、“串装入”以及“串送存”。在形式上它们具有如下一些共同的特点:

① 可处理字节串或字(16 位)串,并在每完成一个字节(或字)的操作后,自动修改指针,去执行下一个字节(或字)的操作;

② 一般情况下,源数据串(源操作数)都存放在数据段,目标串(目标操作数)则存放在附加段。它们在段内的偏移地址则分别由两个变址寄存器 SI 和 DI 指定。只在一些特殊情况下可以通过段重设改变源数据串的段地址;

③ 可以处理的最大串长度为 64K 字节(或字),串长度值由数据寄存器 CX 指定;

④ 多数串操作的指令前可使用重复前缀,以使指令自动重复执行,直到结束;

⑤ 每进行一次操作,地址指针 SI 和 DI 的内容会自动加 1(字操作则加 2)或减 1(减 2)。增加或减少由方向标志位 DF 确定:DF=0,SI 和 DI 为增地址,否则按减地址;

⑥ 在使用重复前缀时,指令还会在执行一次操作后自动修改计数器 CX 的值(CX 减 1)。

串操作指令的执行过程可用图 5.22 所示的流程来表示。

在汇编语言程序设计中,常在串操作指令前添加一个适当的重复操作前缀,以使该指令能重复执行。即指令在执行时不仅能够按照 DF 所决定的方向自动修改地址指针 SI 和 DI 的内容,还可在每完成一次操作后自动修改串长度 CX 的值,重复执行串指令,直至 CX=0 或满足指定的条件为止。在这个过程中,指令指针 IP 会始终保持指向重复前缀(重复前缀本身也是一条指令)的偏移地址。这样,若在重复操作指令执行过程中被外部中断源中断,那么在完成中断处理程

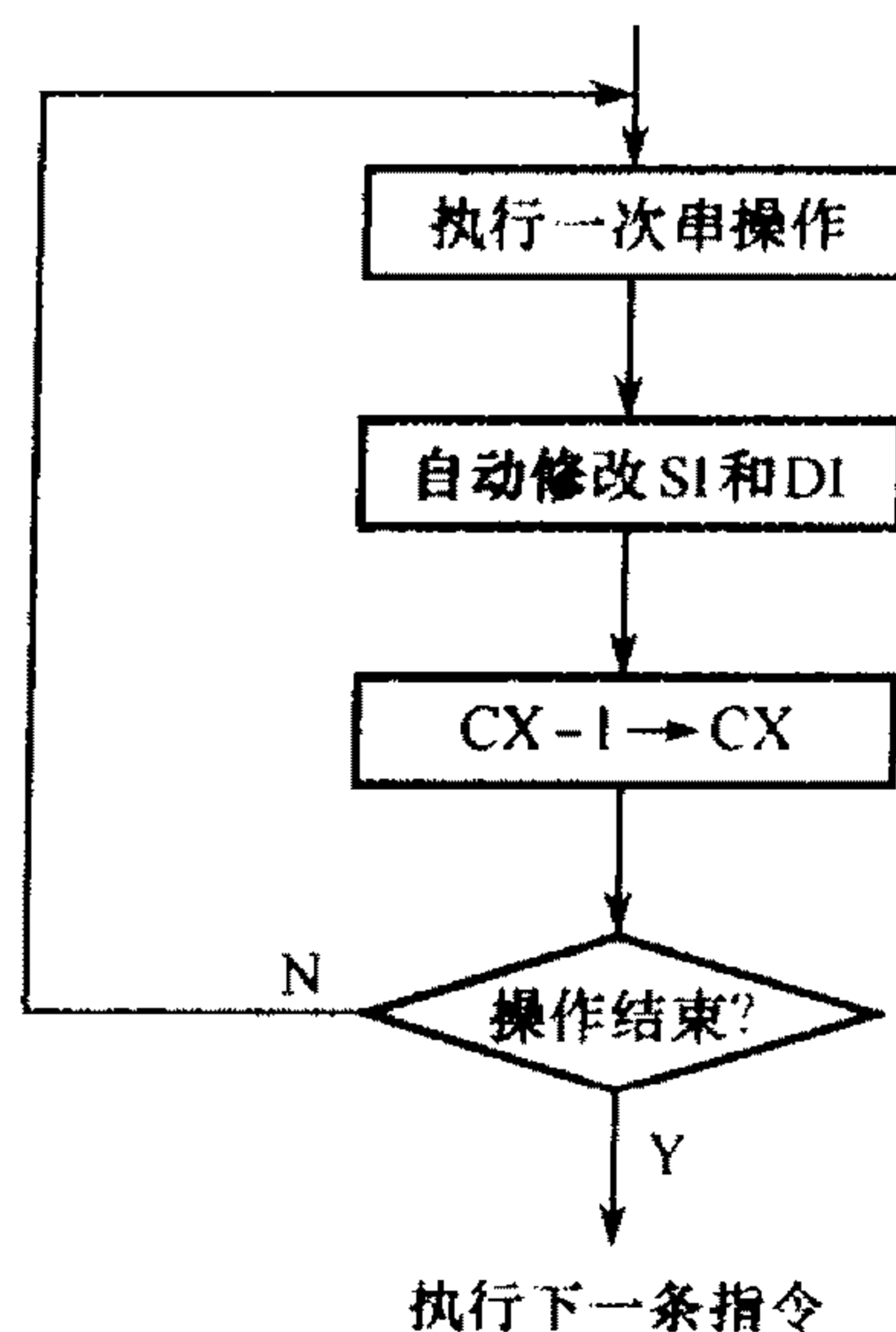


图 5.22 串操作指令的执行过程

序后,将返回继续执行重复操作指令。但在带有多个重复操作前缀时,将禁止 CPU 响应中断。

用于串操作指令的重复操作前缀有 5 条,1 条无条件重复前缀,4 条有条件重复前缀:

- ① REP 无条件重复前缀,即重复执行指令规定的操作,直到 $(CX) = 0$;
- ② REPE 相等时重复,即 $ZF = 1$,且 $(CX) \neq 0$ 时重复;
- ③ REPZ 结果为 0 时重复,即 $ZF = 1$,且 $(CX) \neq 0$ 时重复;
- ④ REPNE 不相等时重复,即 $ZF = 0$,且 $(CX) \neq 0$ 时重复;
- ⑤ REPNZ 结果不为 0 时重复,即 $ZF = 0$,且 $(CX) \neq 0$ 时重复。

综上所述,在汇编语言程序中使用串操作指令时,应预先设置源串地址指针(DS、SI)、目标串地址指针(ES、DI)和重复次数(CX),并通过添加适当的重复操作前缀,简化程序的编写,加快串运算指令的执行速度。

2. 串操作指令

8086CPU 共有 5 条串操作指令,它们是惟一的一组能直接处理源和目标操作数都是存储器单元的指令。

1) 串传送指令(Move String)

串传送指令是将一个数据块从主存储器的某个区域传送到另一个区域。其格式为

```
MOVS OPRD1, OPRD2
MOVSB
MOVSW
```

第一种格式中,OPRD1 为目标串地址,OPRD2 为源串地址。指令将源串地址中的字节或字传送到目标串地址指向的单元中,一般使用默认的段地址,也可通过段重设改变源串的段地址。第二种和第三种格式隐含了两个操作数的地址,此时源串和目标串地址都取默认值,即源串在 DS 段,偏移地址在 SI 中,而目标串在 ES 段,偏移地址在 DI 中。

MOVB 指令一次完成一个字节的传送,MOVSW 一次完成一个字的传送。指令的执行不影响标志位。

串传送指令常与无条件重复前缀 REP 联合使用,以提高程序运行速度。

2) 串比较指令(Compare String)

串比较指令与比较指令 CMP 的操作有点类似,CMP 指令比较的是两个数据,而串比较指令是用于按字或字节逐个比较两个数据块或字符串的大小以及是否相等。指令的格式同样有三种:

```
CMPS OPRD1, OPRD2
CMPSB
CMPSW
```

指令将源串地址中的字节(或字)与目标串地址中的字节(或字)相比较,但比较(相减)结果不送回目标串地址中,而只反映在标志位上。每进行一次比较后自动修改地址指针,指向串中的下一个元素。在以上三种格式中,第一种格式主要用在需要段重设的情况下,至于进行字节比较还是字比较由操作数的类型决定。后两种格式中的 CMPSB 是按字节进行比较,CMPSW 是按字进行比较。

串比较指令与比较指令 CMP 一样会影响标志位,它通常和条件重复前缀 REPE(RepZ)或 REPNE(RepNZ)连用。

3) 串扫描指令(Scan String)

串扫描指令主要用来在一个字符串中搜索特定的关键字。指令将要找的关键字放在累加器 AL(或 AX)中,再与字符串中各字符逐一比较,直到找到字符或是 CX = 0(比较结束)。

指令格式:

```
SCAS OPRD      ;OPRD 为目的串
SCASB
SCASW
```

SCAS 指令与 CMPS 指令执行同样的不回送结果的减法操作,只是这里源操作数为 AL(或 AX)。即用累加器 AL 或 AX 的值减去由 ES:DI 所指定的目标串中的字节或字,相减的结果不改变目标操作数,只影响标志位。同样地,该指令也常和条件重复前缀 REPE(RepZ)或 REPNE(RepNZ)连用。

4) 串装入指令(Load String)

指令格式:

```
LODS OPRD      ;OPRD 为源串
LODSB
LODSW
```

LODS 指令把由 DS:SI 指向的源串中的字节或字取到累加器 AL 或 AX 中,并在这之后根据 DF 的值自动修改指针 SI,以指向下一个要装入的字节或字。

LODS 指令不影响标志位,且一般不带重复前缀,因为每重复一次,AL 或 AX 中内容将被后一次所装入的字符所取代。

LODSB 指令可用来代替以下两条指令:

```
MOV AL,[SI]
INC SI
```

而 LODSW 指令可用来代替以下 3 条指令:

```
MOV AX,[SI]
INC SI
```


INC SI

5) 串存储指令 (Store String)

指令格式:

STOS OPRD ;OPRD 为目标串

STOSB

STOSW

STOS 指令与 LODS 指令执行相反的操作,它将累加器 AL 中的字节或 AX 中的字存到由 ES:DI 指向的存储器单元中,并在这之后根据 DF 的值自动修改指针 DI 的值(增量或减量),以指向下一个存储单元。利用重复前缀 REP,可对连续的存储单元存入相同的值。指令对标志位没有影响。

5.3.5 程序控制指令

这类指令主要用于控制程序流的转移。在多数情况下,计算机是按顺序方式执行程序,但有时也会需要脱离这种顺序转移到另一个程序段去执行,或者是循环地执行某一个程序段。

按照其工作性质,这类指令可分为:转移指令、循环控制指令、过程调用指令和中断控制指令四大类。

1. 转移指令

转移指令包括无条件的转移指令和有条件的转移指令两类。

1) 无条件转移指令 JMP(Jump)

顾名思义,无条件转移指令可不受任何条件的约束而将程序转移到规定的目标地址,并从该地址开始继续程序的执行。目标地址既可以在当前代码段内,也可以在不同的代码段。对应于 5.2 节中寻找转移地址的寻址方式,JMP 指令具有如下几种格式:

(1) 段内直接转移

指令格式:

JMP LABEL

指令直接给出转移目的地的符号地址 LABEL。该指令可实现程序在当前代码段 - 32 768 ~ + 32 767 的地址范围内的转移。

(2) 段内间接转移

指令格式:

JMP OPRD

这里的操作数 OPRD 是 16 位的寄存器或者存储器地址。指令的执行是用 OPRD 指定的

16 位寄存器内容或存储器两单元内容作为目标的偏移地址来代替原来 IP 的内容,从而实现程序的转移。例如:

JMP BX ;本指令执行后, $\langle IP \rangle = (BX)$

JMP WORD PTR[BX + DI]

对上面第二条指令,设指令执行前: $(DS) = 3000H$, $(BX) = 1300H$, $(DI) = 1200H$, $(32500H) = 2350H$,则指令执行后, $(IP) = 2350H$,指令执行的过程如图 5.23 所示。

由于是段内转移,其范围一定在当前代码段内,所以 CS 的内容不变。

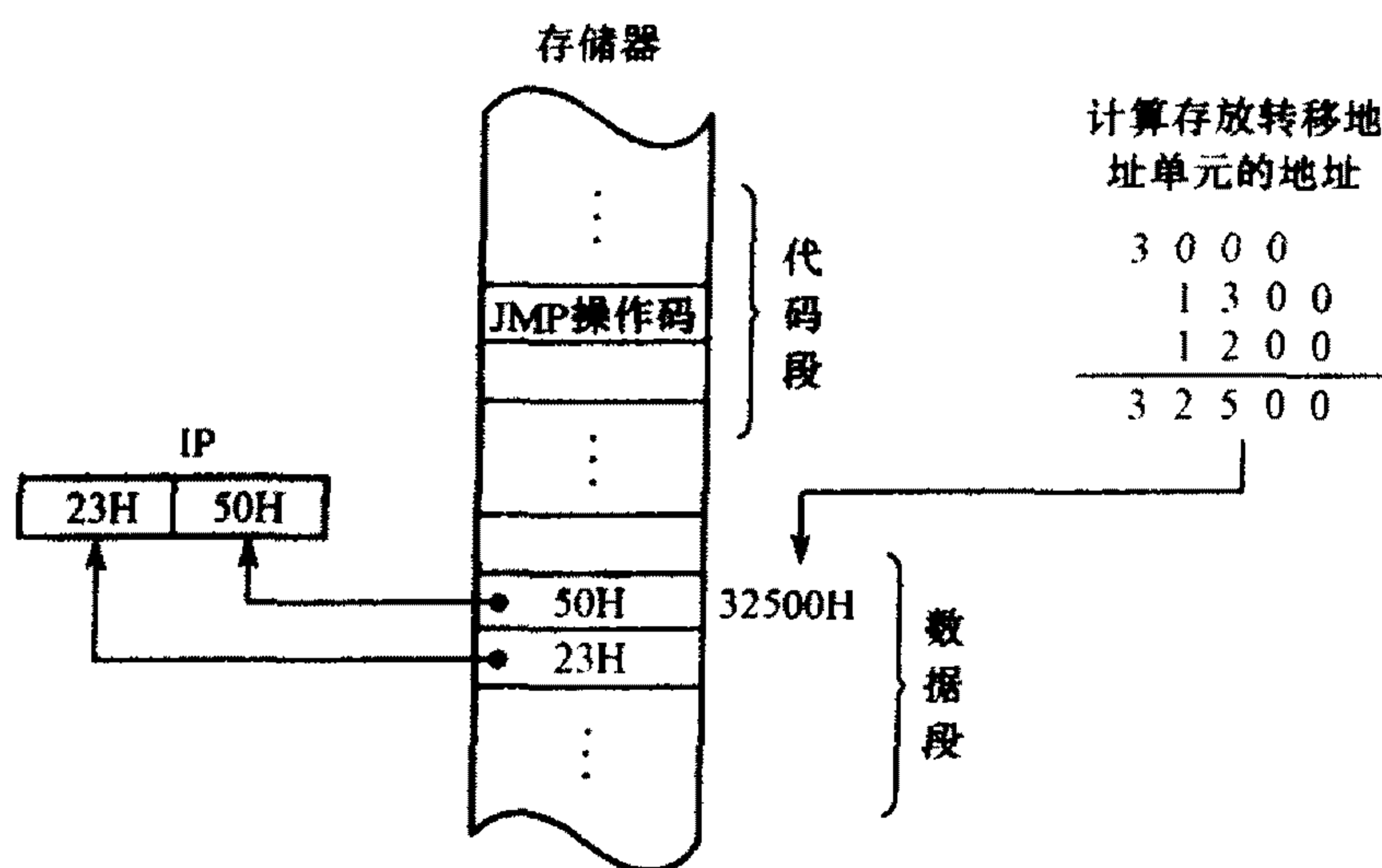


图 5.23 段内间接转移指令操作示意图

(3) 段间直接转移

指令格式:

JMP FAR LABEL

这里的 FAR 表明其后的标号 LABEL 是一个远标号,即它在另一个代码段内。该指令使程序转移到另一个代码段(CS:IP)继续执行,如图 5.24 所示。指令操作码后的 4 个字节的内容是由汇编程序根据 LABEL 的位置确定的。例如:

JMP FAR PTR NEXT ;远转移到 NEXT 处

JMP 8000H:1200H ; $(IP) \leftarrow 1200H$, $(CS) \leftarrow 8000H$

(4) 段间间接转移

指令格式:

JMP OPRD

这里,操作数 OPRD 是一个 32 位的存储器地址。指令的执行是将指定的连续四个内存单元的内容送入 IP 和 CS(低字的内容送 IP,高字的内容 CS),从而程序转移到另一个代码段

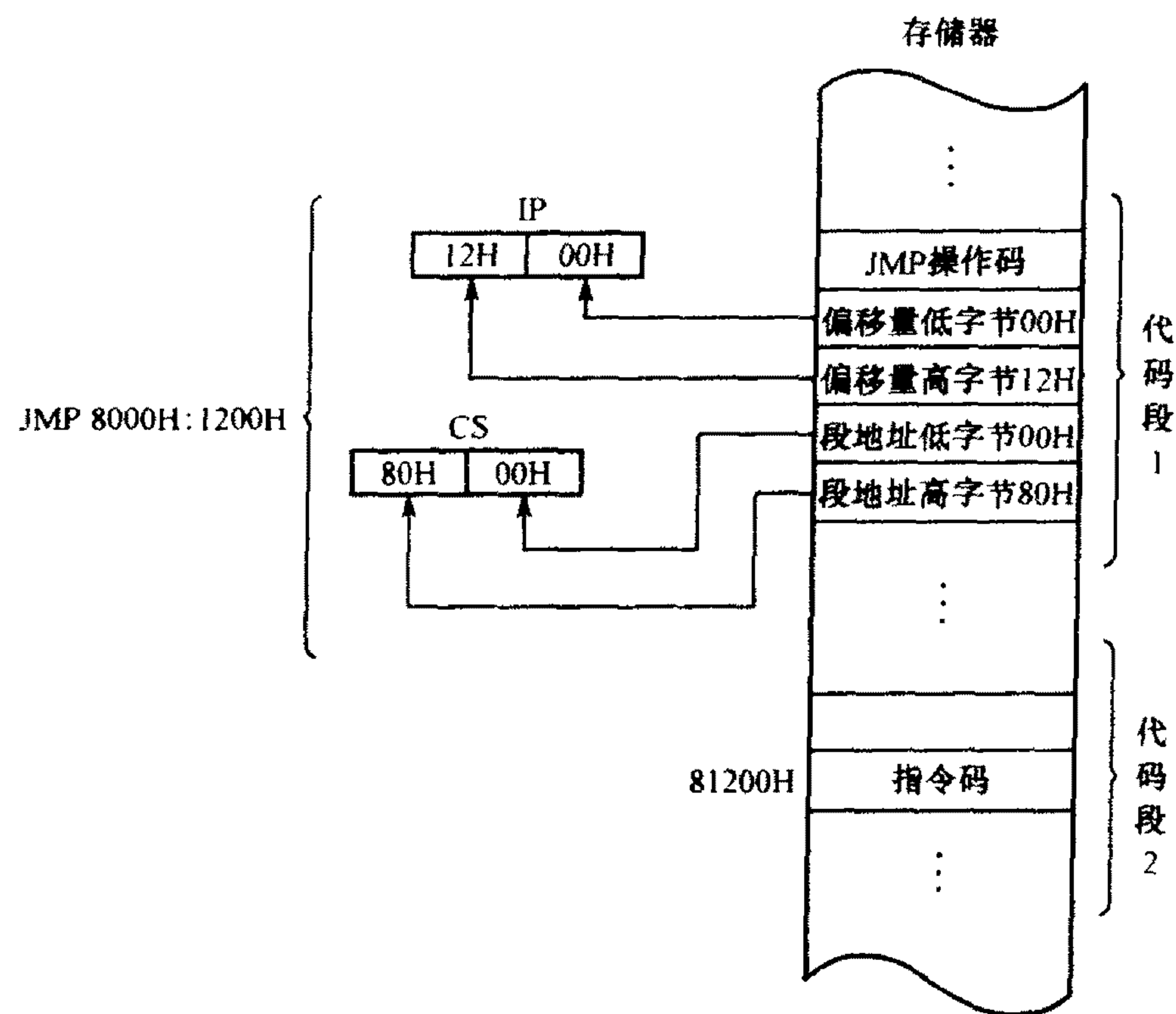


图 5.24 段间直接转移操作示意图

继续执行。此处的存储单元地址可采用本章前面讲过的各种寻址方式(立即数和寄存器方式除外)。

例 5-24

JMP DWORD PTR[BX]

设指令执行前:

(DS) = 3000H, (BX) = 3000H, (33000H) = 0BH, (33001H) = 20H, (33002H) = 10H,
(33003H) = 80H

则指令执行后:

(IP) = 200BH, (CS) = 8010H

转移的目标地址 = 8210BH。其操作示意如图 5.25 所示。

由于段间转移是控制程序转移到另一个代码段中,不仅 IP 的内容要改变,CS 的内容也要改变,即转移地址一定是 32 位字长。因此,在操作数前要加上 DWORD PTR,表示其后的操作数为双字。JMP 指令对标志位无影响。

2) 条件转移指令 JCC

8086 具有 18 条不同的条件转移指令(见表 5-6)。条件转移指令是根据计算机处理的结果来决定程序下一步如何执行。它先测试前一条指令执行后标志位的状态(转移的条件

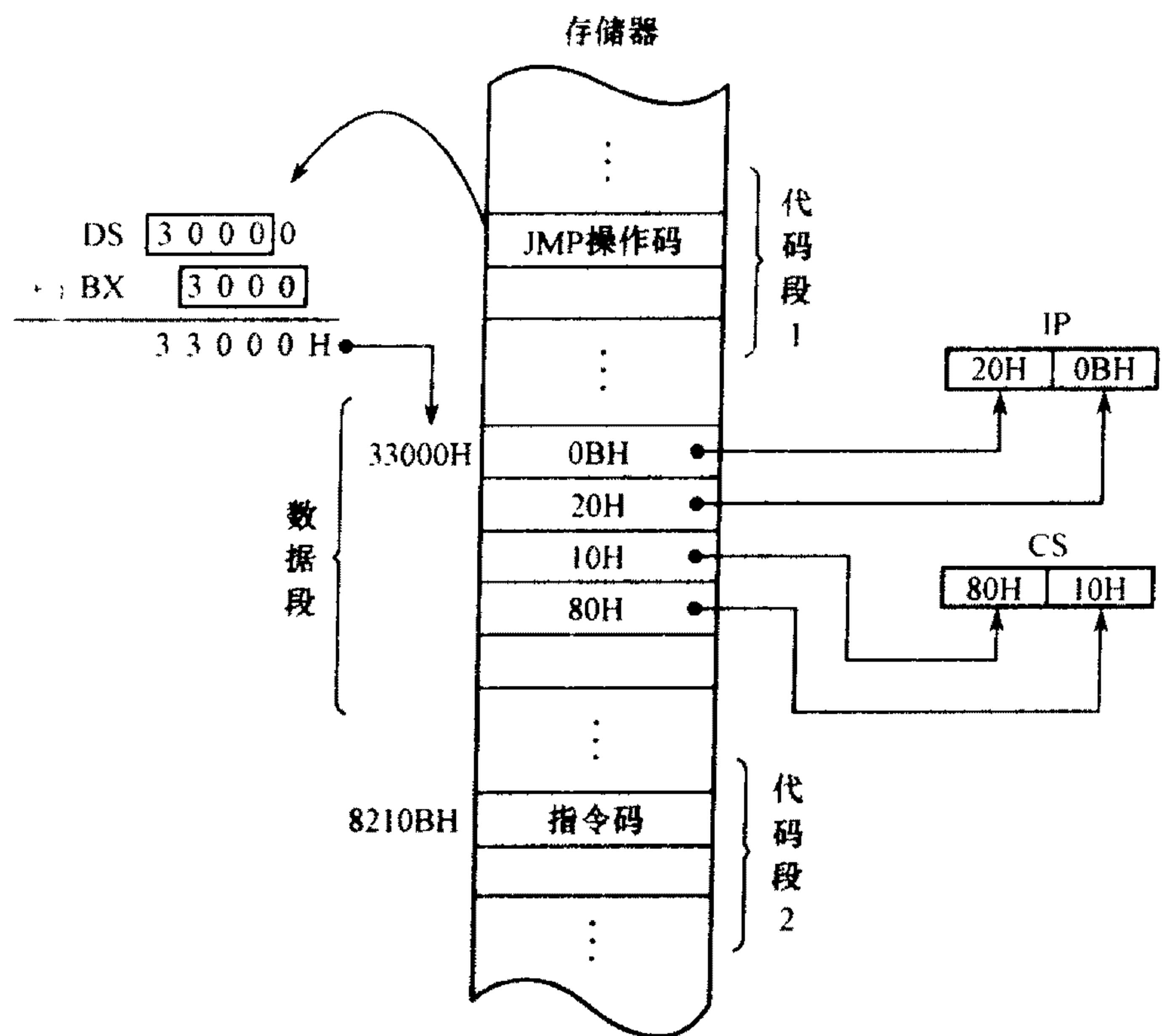


图 5.25 段间间接转移操作示意图

码),再根据测试结果决定程序是否转移。若条件为“真”,则程序转移到指令指定的地址去继续执行;若条件为“假”,则顺序执行下一条指令。所有的条件转移都是直接寻址方式的短转移,即只能在以当前 IP 内容为中心的 $-128 \sim +127$ 字节范围内转移。条件指令不影响标志位。

表 5-6 条件转移指令

指令名称	汇编格式	转移条件	备注
CX 内容为 0 转移	JCXZ target	$(CX) = 0$	
大于/不小于或等于转移	JG/JNLE target	$(SF) = (OF)$ 且 $(ZF) = 0$	带符号数
大于或等于/不小于转移	JGE/JNL target	$(SF) = (OF)$	带符号数
小于/不大于或等于转移	JL/JNGE target	$(SF) \neq (OF)$ 且 $(ZF) = 0$	带符号数
小于或等于/不大于转移	JLE/JNG target	$(SF) \neq (OF)$ 或 $(ZF) = 1$	带符号数
溢出转移	JO target	$(OF) = 1$	
不溢出转移	JNO target	$(OF) = 0$	
结果为负转移	JS target	$(SF) = 1$	
结果为正转移	JNS target	$(SF) = 0$	

续表

指令名称	汇编格式	转移条件	备注
高于/不低于或等于转移	JA/JNBE target	(CF) = 0 且 (ZF) = 0	无符号数
高于或等于/不低于转移	JAE/JNB target	(CF) = 0	无符号数
低于/不高于或等于转移	JB/JNAE target	(CF) = 1	无符号数
低于或等于/不高于转移	JBE/JNA target	(CF) = 1 或 (ZF) = 1	无符号数
进位转移	JC target	(CF) = 1	
无进位转移	JNC target	(CF) = 0	
等于或为零转移	JE/JZ target	(ZF) = 0	
不等于或非零转移	JNE/JNZ target	(ZF) = 1	
奇偶校验为偶转移	JP/JPE target	(PF) = 1	
奇偶校验为奇转移	JP/JPO target	(PF) = 0	

条件转移指令的条件码的建立和转移的判断一般需要两条指令完成。由转移指令前边的指令建立转移的条件,通常都是算术运算指令,再根据条件是否成立决定程序是否转移。

在使用条件转移指令时,应首先确定要转移的条件,然后再根据具体情况选用不同的指令。

2. 循环控制指令

在实际的程序设计中,有时需要连续重复地执行某个程序段,这样的程序段称为循环程序。循环控制指令就是控制程序循环的。指令控制程序在以当前 IP 内容为中心的 -128 ~ +127 范围内循环执行。循环的次数由 CX 寄存器指定。一般情况下,循环控制指令放在循环程序的开始或结尾,每循环一次,CX 内容减 1,若 (CX) ≠ 0,则继续循环;否则就退出循环。

循环控制指令包括无条件循环和条件循环两类共 3 条指令。

1) 无条件循环指令 LOOP

指令格式:

LOOP LABEL

这里的 LABEL 为近地址标号。指令的执行是先将 CX 内容减 1,再判断 CX 是否为零,若 CX 不为 0,则转至目标地址继续循环;否则退出循环,执行下一条指令。

LOOP 指令相当于以下两指令的组合:

DEC CX

JNZ NEXT

2) 条件循环指令

条件循环指令共有两条,分别为:

(1) LOOPZ(或 LOOPE)指令

指令格式:

LOOPZ LABEL

或

LOOPE LABEL

这条指令使程序继续循环的条件是 $CX \neq 0$, 且 $ZF = 1$; 若 $CX = 0$ 或者 $ZF = 0$, 则退出循环。

(2) LOOPNZ(或 LOOPNE)指令

指令格式:

LOOPNZ LABEL

或

LOOPNE LABEL

这条指令与 LOOPZ 指令类似, 只是其中 ZF 条件与之相反。当 $CX = 0$ 且 $ZF = 0$ 时转至目标地址继续循环, 否则退出循环。

3 条循环指令的执行对标志位均不影响。

3. 过程调用和返回

在编程过程中, 为了节省存储空间并简化程序设计, 常常将一些具有特定功能、需经常使用的程序段编写成一个独立的程序模块, 在需要时可随时调用, 而不必再多次重复编写。这样的程序模块称之为子程序或过程。

8086 指令系统为实现调用子程序这一功能提供了过程调用指令 CALL 和返回指令 RET。CALL 指令与 JMP 指令。转移指令在转移到目标地址后就从新的地址开始一直执行下去, 而过程调用指令则是在子程序执行结束后还要再返回到原调用程序, 继续执行 CALL 指令的下一条指令。所以, 过程调用指令需要在转入子程序前保护返回地址, 保护的方法是将返回地址压入堆栈中。另外, 在必要时还要保护某些通用寄存器及状态寄存器的内容。这部分工作通常可在子程序中完成。

返回指令 RET 是和调用指令 CALL 配对使用的指令, 它负责将返回地址从堆栈中弹出, 以便继续执行 CALL 指令的下一条指令。

调用指令 CALL 的执行过程如下:

- ① CPU 将下 CALL 的下一条指令的地址(称为返回地址, 或称断点)压入堆栈保存;
- ② 将子程序入口地址(即子程序第 1 条指令的地址)赋给 IP(或 CS 和 IP), 开始执行子程序;
- ③ 必要时保护某些通用寄存器及状态寄存器的内容, 并在子程序执行结束后恢复;
- ④ 将栈顶保留的返回地址弹出到 IP(或 CS 和 IP), 继续执行原调用程序。

由于子程序有可能与主程序同在一个代码段内, 也有可能不在同一个段, 所以与无条件转移指令一样, CALL 指令也有四种形式, 即段内直接调用、段内间接调用、段间直接调用以及段间间接调用。

1) 段内直接调用

指令格式:

CALL NEAR PROC

这里的 PROC 是一个近过程的符号地址,表示指令调用的过程是在当前代码段内。指令在汇编后,会得到 CALL 指令的下一条指令与被调用过程的入口地址之间相差的 16 位相对位移量(也可以理解为是字节表示的距离)。

CALL 指令执行时,首先将下面一条指令的偏移地址压入堆栈,然后将指令中 16 位的相对位移量和当前 IP 的内容相加,新的 IP 内容即为所调用过程的入口地址(确切地说是入口地址的偏移地址)。执行过程表示如下:

$$(SP) \leftarrow (SP) - 2$$

$$(SP) + 1 \leftarrow (IP_H)$$

$$(SP) \leftarrow (IP_L)$$

$$(IP) \leftarrow (IP) + 16 \text{ 位偏移量}$$

对于段内调用,指令中的 NEAR 可以省略。例如“CALL TIME”指令将调用一个名为 TIME 的近过程。

2) 段内间接调用

指令格式:

CALL OPRD

这里的 OPRD 为 16 位寄存器或两个存储器单元的内容。它的内容就是一个近过程的入口地址。指令的操作是将 CALL 指令的下面一条指令的偏移地址压入堆栈,然后将 OPRD 的内容(16 位通用寄存器或存储器两单元)送指令指针 IP。

例 5-25

CALL AX ; (IP) ← (AX), 子程序的入口地址由 AX 给出

CALL WORD PTR[BX] ; (IP) ← ((BX + 1): (BX)), 子程序的入口地址为

; BX 和 BX + 1 两存储单元的内容

若 (DS) = 8000H, (BX) = 1200H, 指令操作示意图如图 5.26 所示。

3) 段间直接调用

指令格式:

CALL FAR PROC

这里的 PROC 是一个远过程的符号地址,表示指令调用的过程在另外的代码段内。获取子程序的入口地址的方法与 JMP 指令的段间直接转移中转移地址的获取方法相同。只是这里需要在进入子程序之前要先将 CALL 指令的下一条指令的地址,即 CS 和 IP 寄存器的内容压入堆栈保存,然后再用指令中给出的段地址取代 CS 的内容,偏移地址取代 IP 的内

容。执行过程如下：

$(SP) \leftarrow (SP) - 2, ((SP) + 1 : (SP)) \leftarrow (CS), (CS) \leftarrow$
 \leftarrow 所调用过程入口的段地址；
 $(SP) \leftarrow (SP) - 2, ((SP) + 1 : (SP)) \leftarrow (IP), (IP) \leftarrow$
 \leftarrow 所调用过程入口的偏移地址。

例如，指令“CALL 3000H:2100H”直接给出了所要调用的过程的段地址和偏移地址“3000H:2100H”。

4) 段间间接调用

指令格式：

CALL OPRD

与无条件转移指令一样，这里的 OPRD 为 32 位的存储器地址。指令的操作是将 CALL 指令的下一条指令地址，即 CS 和 IP 的内容压入堆栈，然后把指令中指定的连续 4 个存储单元中内容送 IP 及 CS，低地址的两个单元内容为偏移地址，送入 IP；高地址的两个单元内容为段地址，送入 CS。

例 5-26

CALL DWORD PTR[SI] ;调用的入口地址为(SI+0)~(SI+3)的存储单元的内容
 设(DS) = 6000H, (SI) = 0560H, 指令操作的示意如图 5.27 所示。

5) 返回指令 RET

返回指令执行与调用指令执行相反的操作。对于近过程(与原调用程序在同一段内)，用 RET 返回原调用程序时，只需从堆栈顶部弹出一个字的内容给 IP 作为返回的偏移地址。对于远过程(与原调用程序不在同一段)，用 RET 返回原调用程序时，则需从堆栈顶部弹出两个字作为返回地址，先弹出一个字的内容给 IP 作为返回的偏移地址，再弹出一个字的内容给 CS 作为返回的段地址。

无论是段间返回还是段内返回，返回指令在形式上都是 RET。

返回指令一般作为子程序的最后一条语句。所有的返回指令都不影响标志位。

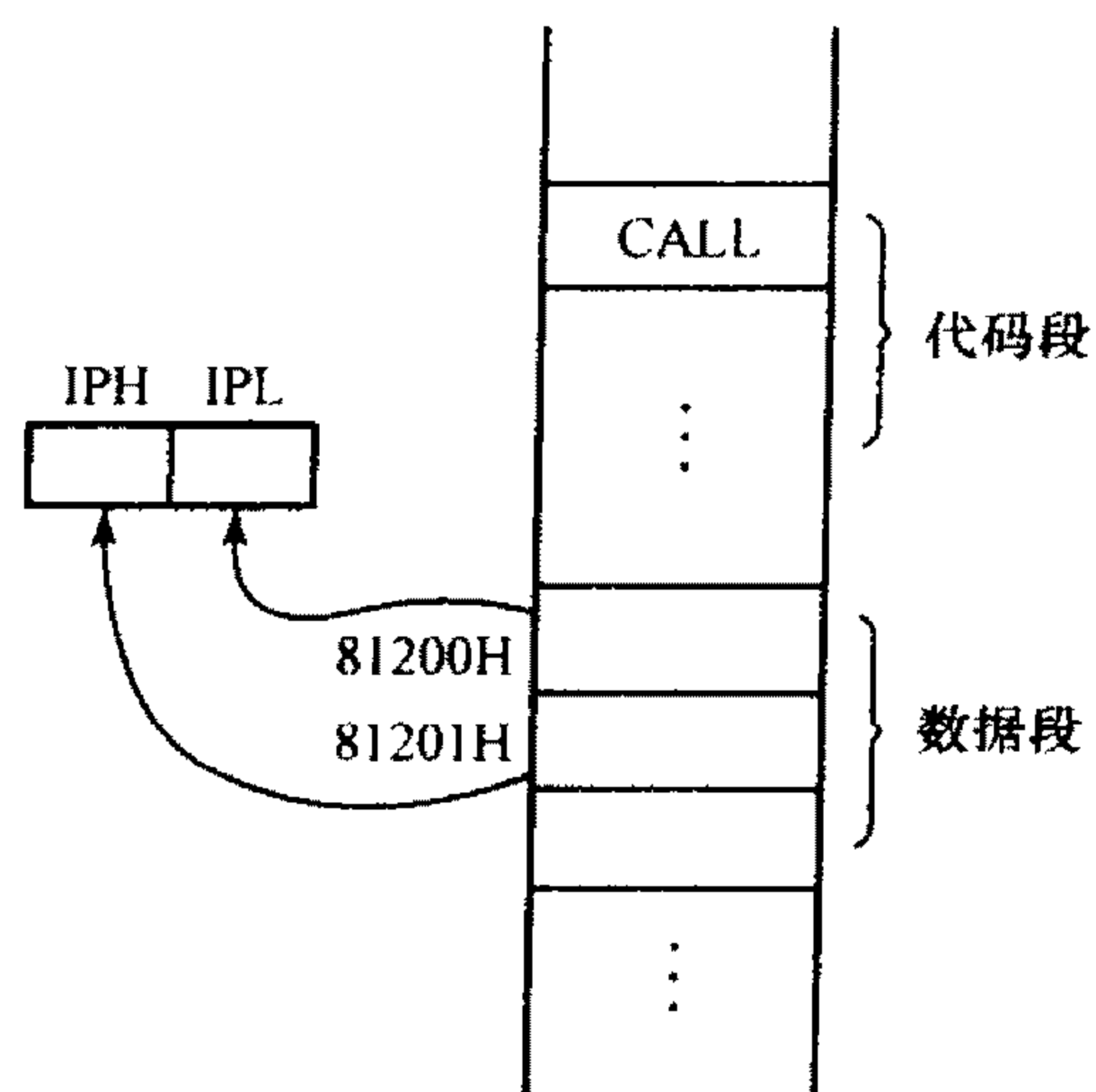


图 5.26 段内间接调用指令操作示意图

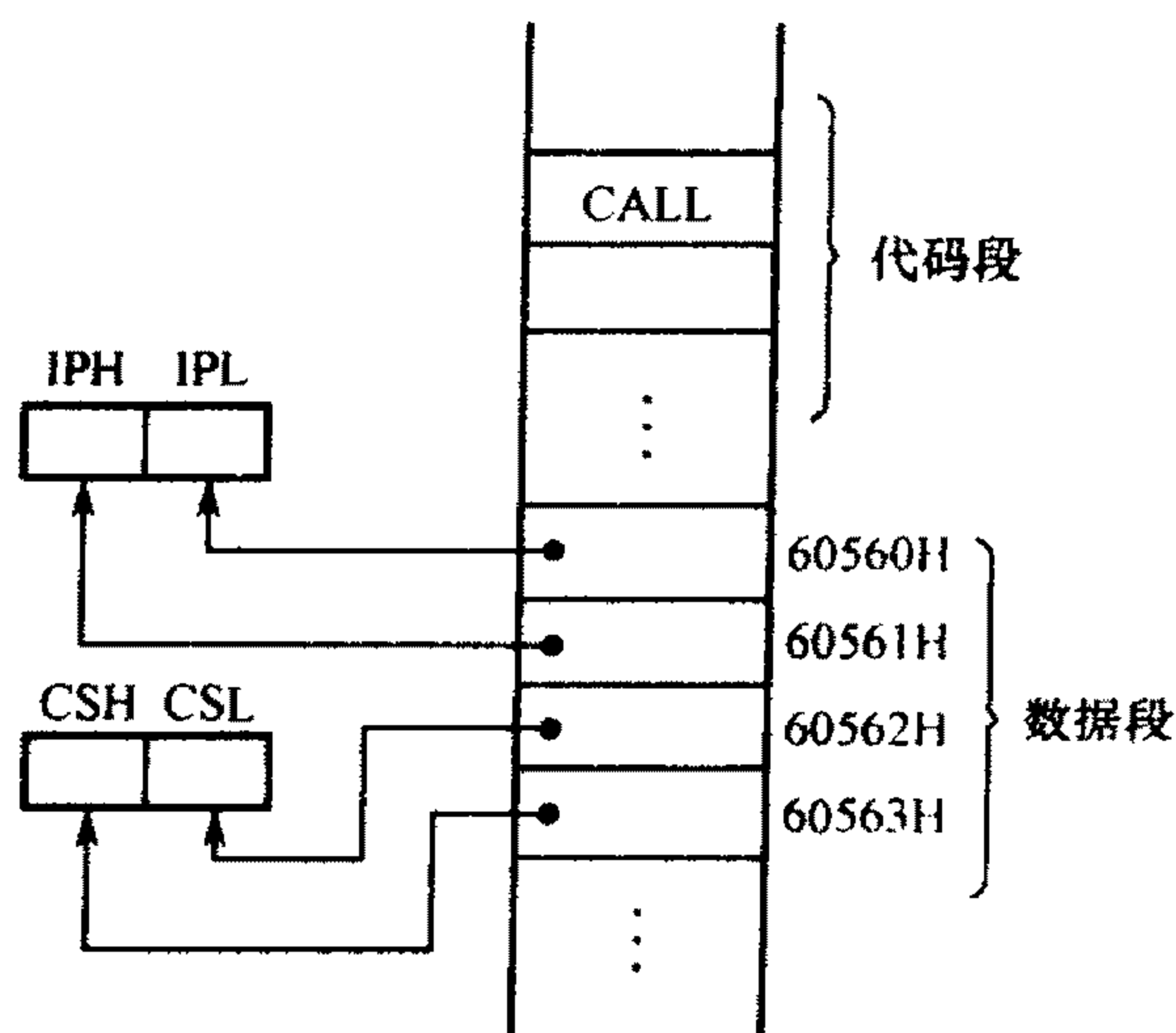


图 5.27 段间间接调用指令操作示意图

3. 中断指令

在程序运行期间,有时会产生一些随机事件,如运算产生溢出、请求数据传送等;另外,在计算机的运行过程中,也可能会发生如输入/输出设备故障、存储器校验出错等意外情况。这些都会使得计算机不得不暂时中止正在运行的程序,而转去执行一组相应的处理程序,处理完毕后又返回原被中止的程序并继续执行,这样一个过程称为中断。

8086 中断系统分为外部中断(或叫硬件中断)和内部中断(或叫软件中断)。外部中断主要用来处理外设和 CPU 之间的通信,内部中断主要指中断指令引起的中断。

中断指令用于产生软件中断,以执行一段特殊的中断处理过程。中断指令主要有以下几个用途:

① 系统功能调用 为了便于各种程序设计,操作系统为用户提供了一些通用的功能子程序,如控制台输入/输出、文件读/写、软硬件资源管理、通信等(参见附录 C)。用户程序可以通过中断指令(如 INT 21H)直接调用这些服务子程序,而不用再自己编写,从而大大简化了应用软件开发;

② 用于实现一些特殊功能 如调试程序时单步运行、断点等;

③ 调用 BIOS 提供的硬件底层服务。

有关中断的详细论述将在本书第 7 章进行,这里仅介绍 8086 指令系统中的 3 条软件中断的指令的格式及操作。

1) INT 指令

指令格式:

INT n

n 为中断向量码(也称中断类型码),取值范围为 0 ~ 255。

例如:INT 21H 即表示执行中断向量码为 21H 的指令。

指令执行时,CPU 根据 n 的值计算出中断向量的地址,然后从该地址中取出中断服务程序的入口地址,并转到该中断服务子程序去执行。中断向量地址的计算方法是将中断向量码 n 乘 4。INT 指令的具体操作步骤如下:

① $(SP) \leftarrow (SP) - 2, ((SP) + 1 : (SP)) \leftarrow (FLAGS)$

把标志寄存器的内容压入堆栈

② $(TF) \leftarrow 0, (IF) \leftarrow 0$

清除 IF 和 TF,以保证在中断服务子程序中不会被再次中断,并且也不会响应单步中断

③ $(SP) \leftarrow (SP) - 2, ((SP) + 1 : (SP)) \leftarrow (CS)$

$(SP) \leftarrow (SP) - 2, ((SP) + 1 : (SP)) \leftarrow (IP)$

把断点地址(INT 指令的下一条指令地址)的段地址和偏移地址压入堆栈

$$\textcircled{4} \quad (\text{IP}) \leftarrow ((n \times 4) + 1 : ((n \times 4)))$$

$$(\text{CS}) \leftarrow ((n \times 4) + 3 : ((n \times 4) + 2))$$

由 $n \times 4$ 得到中断向量地址,并进而得到中断处理子程序的入口地址

以上操作完成后,CS:IP 就指向中断服务程序的第一条指令,此后 CPU 开始执行中断服务子程序。

INT n 指令除复位 IF 和 TF 外,对其他标志无影响。

从 CPU 执行中断指令的过程可以看出,INT 指令的基本操作与存储器寻址的段间间接调用指令非常相像,所不同的是:

- INT 指令除保存断点外还要将标志寄存器 FLAGS 压入堆栈保存,而 CALL 指令则不必;
- INT 影响 IF 和 TF 标志,而 CALL 指令不影响;
- 中断服务程序入口地址放在内存的固定位置,以便通过中断向量号找到它。而 CALL 指令可任意指定子程序入口地址的存放位置。

2) 溢出中断 INTO

前面提到,有符号数运算中的溢出是一种错误,在程序中应尽量避免(如果避免不了,也希望能及时发现,否则程序再往下运行,其结果便毫无意义了),为此 8086 指令系统专门提供了一条溢出中断指令,用来判断有符号数加减运算是否溢出。一般使 INTO 指令紧跟在有符号数加、减运算指令的后面。若运算使 $\text{OF} = 1$,则 INTO 指令会使控制自动转到溢出中断处理程序,进行溢出处理。若 $\text{OF} = 0$,则 INTO 指令不执行任何操作,使程序继续执行下一条指令。INTO 指令从检查到 $\text{OF} = 1$ 到转移至溢出中断处理程序的执行过程与 INT n 指令是类似的,只不过 INTO 指令是 $n = 4$ 的 INT 指令,其向量地址为 0010H。即 INTO 指令与 INT 4 指令调用的是同一个中断服务程序。

3) 中断返回指令 IRET

中断返回指令 IRET 用于从中断服务子程序返回到被中断处继续执行原程序。任何中断服务子程序无论是由外部中断引起的,还是内部中断引起的,其最后一条指令都是 IRET 指令。该指令首先将堆栈中的断点地址弹出到 IP 和 CS,接着将 INT 指令执行时压入堆栈的标志字弹出到标志寄存器,以恢复中断前的标志状态。显然本指令对各标志位均有影响。指令的操作为:

$$(\text{IP}) \leftarrow ((\text{SP}) + 1 : (\text{SP})), (\text{SP}) \leftarrow (\text{SP}) + 2$$

$$(\text{CS}) \leftarrow ((\text{SP}) + 1 : (\text{SP})), (\text{SP}) \leftarrow (\text{SP}) + 2$$

$$(\text{FLAGS}) \leftarrow ((\text{SP}) + 1 : (\text{SP})), (\text{SP}) \leftarrow (\text{SP}) + 2$$

5.3.6 处理器控制指令

处理器控制指令包括对标志位的操作及对 CPU 本身的控制两大类,见表 5-7。

表 5-7 处理器控制指令

汇编格式		操作
标志位操作指令	CLC	$CF \leftarrow 0$;清进位标志位
	STC	$CF \leftarrow 1$;进位标志位置位
	CMC	$CF \leftarrow \overline{CF}$;进位标志位取反
	CLD	$DF \leftarrow 0$;清方向标志位,串操作从低地址到高地址
	STD	$DF \leftarrow 1$;方向标志位置位,串操作从高地址到低地址
	CLI	$IF \leftarrow 0$;清中断标志位,即关中断
	STI	$IF \leftarrow 1$;中断标志位置位,即开中断
外部同步指令	HLT	暂停指令
	WAIT	等待指令
	ESC	处理器交权指令
	LOCK	总线封锁指令
	NOP	空操作指令

1. 标志操作指令

对标志位的操作指令共有 7 条,包括 3 条针对进位标志位的操作指令,2 条针对方向标志位的操作指令和 2 条对中断标志位的操作指令。各指令的含义见表 5-6。

2. 针对处理器操作的指令

这类指令主要用于 CPU 和协处理器或外部设备的协调和同步控制,所以也称为外部同步指令。

1) 暂停指令 HLT

执行暂停指令将使 CPU 处于暂停状态,只有在重新启动或有外部中断发生时,才退出暂停。这条指令常用来等待中断的产生。

2) 空操作指令 NOP

这条指令的执行不产生任何结果,仅消耗 3 个时钟周期的时间,常用于程序的延时等。

3) 等待指令 WAIT

8086 每隔 5 个时钟周期都要检查一次 \overline{TEST} 引脚,当该引脚上的信号为高电平(无效)时,执行 WAIT 指令会使 CPU 进入等待状态。而一旦检测到 \overline{TEST} 引脚上的信号变为低电平,

便退出等待状态。

该指令主要用于 8086 与协处理器和外部设备的同步,也可用来等待被允许的外部中断的到来,但中断任务执行完后仍返回到 WAIT 指令,继续执行等待状态。

此指令对标志位无影响。

4) 总线锁定指令 LOCK

LOCK 指令是一个前缀,可放在任何一条指令前面,它主要是为多机共享资源而设计的。这条指令的执行可使 8086 的 LOCK 引脚低电平有效,从而使得加有 LOCK 前缀的指令在执行期间封锁外部总线,不允许其他处理器工作,而只能使某个处理器工作,以免多机共享资源情况下,出现不正确使用内存的情况。这个过程会一直持续到该指令执行结束。本指令不影响标志位。

5) 处理器交权指令 ESC

当 8086 工作在最大模式下时,可以配备协处理器,如 8087,共同构成多处理器系统。8087 协处理器具有较强的浮点运算功能,可使系统的运算速度大大提高。在这种多处理器系统中,当 8086 需要 8087 配合时,就在程序中加一条 ESC 指令,使一部分指令转到协处理器上执行。该指令的一般格式为

ESC OPRD

这里,OPRD 是一个存储器操作数。指令执行时,把一个指定的存储单元的内容送到数据总线上,由协处理器获取后,完成相应的操作。

本指令不影响标志位。

5.4 80X86 新增指令及汇编语言源程序结构

上一节所介绍的 8086/8088 指令系统,也是 80X86 系列 CPU 的基本指令系统,它的指令编码、寻址方式与 Intel 系列 CPU 运行在实地址模式下是完全相同的。由于从 80386 起增加了虚地址模式,因此相应地增加了虚地址模式下的寻址方式,其指令系统也随之扩充,功能进一步增强。本节以 80386 为例,简要介绍 80X86CPU 在 8086 指令系统基础上新增的指令及寻址方式。

5.4.1 80X86 虚地址下的寻址方式

相对于实地址模式下的八种对操作数的寻址方式,80X86 增加了虚地址下寻址 32 位数的寻址能力。

1. 立即寻址

这里的立即数可以是 8 位、16 位或 32 位。如：

MOV EAX,12345678H ;将 32 位立即数送 32 位寄存器 EAX

2. 直接寻址

由指令直接给出 8 位、16 位或 32 位的偏移地址，如：

MOV EAX,[11202020H]

3. 寄存器寻址

存放操作数的通用寄存器可控制为 32 为。例：

MOV EAX,EBX

4. 寄存器间接寻址

与 8086 不同,80X86 允许所有的通用寄存器都可用作间接寻址。除 ESP 和 EBP 默认数据在堆栈段外,其他通用寄存器作间址寄存器时,都默认数据在数据段。但允许段重设。

例：

MOV EBX,[EAX] ;将数据段中偏移地址为 EAX 内容的 4 个字节数送 EBX

MOV EAX,[BX] ;将数据段中偏移地址为 BX 内容的 4 个字节数送 EAX

MOV AX,[EBP] ;将堆栈段中偏移地址为 EBP 内容的 2 个字节数送 AX

5. 寄存器相对寻址

与 8086 的寻址方式类似,这种寻址方式下操作数的偏移地址由寄存器的内容加上一个位移量构成。在偏移量是 32 位的情况下,操作数的偏移地址由任何一个通用寄存器的内容加上一个 32 位位移量构成。除 ESP 和 EBP 默认数据在数据段之外,其余都默认在数据段。

例：MOV AX,DATA[EBX]

MOV EBX,DATA[EBP]

这里的 DATA 为 32 位位移量。

6. 基址、变址寻址

操作数的偏移地址等于基址寄存器的内容加上变址寄存器的内容。在 32 位偏移量的情况下,基址寄存器和变址寄存器可以是任何一个通用寄存器。由基址寄存器决定数据默认在哪个段。例：

MOV EAX,[EBX][ESI] ;默认数据在数据段

MOV EBX,[EBP][EDI] ;默认数据在堆栈段

7. 基址、变址、相对寻址

此时,操作数的偏移地址等于基址寄存器的内容加上变址寄存器的内容,再加上位移

量。同样,当位移量是 32 位时,基址寄存器和变址寄存器可以是任何一个通用寄存器。例:

```
MOV EAX,[EBX + EDI + 0FFFFFF0H]
```

8. 带比例因子的寻址方式

1) 带比例因子的变址寻址

这种寻址方式是将变址寄存器的内容乘上一个比例因子,再加上一个位移量形成存放操作数的有效偏移地址。比例因子的选取与操作数的字长相同,如操作数可以是 1 字节、2 字节、4 字节或 8 字节,相应地,比例因子可以是 1、2、4 或 8。例:

```
MOV EAX,DATA[ESI×4]
```

2) 带比例因子的基址、变址寻址

该方式下操作数的偏移地址等于某个通用寄存器的内容乘以比例因子,再加上另一个通用寄存器内容。乘比例因子的那个寄存器被认为是变址寄存器,操作数默认的段由选用的基址寄存器决定,如果是 ESP 或 EBP,则数据默认在堆栈段。例:

```
MOV EBX,[EDX×4][EAX]      ;默认数据在数据段
```

```
MOV EAX,[EBX×4][EBP]      ;默认数据在堆栈段
```

```
MOV AX,[EBP×2][EBX]       ;默认数据在数据段
```

3) 带比例因子的基址、变址、相对寻址

此时操作数的偏移地址由基址寄存器的内容加上变址寄存器的内容乘以比例因子,再加上一个位移量构成。例:

```
MOV EAX,[EBX + DATA][EDI×4]
```

5.4.2 80X86 新增指令

随着 Intel 公司系列微处理器技术的发展,CPU 的字长由 16 位扩展到了 32 位,其指令系统也随之得到了相应的扩充和增强。除增强了部分 8086 指令的功能外,还增加了一些新的指令,以使程序的编写更加方便,并使整个系统的功能和执行速度提高。

80286 属于高档的 16 位处理器,它在 8086 指令系统的基础上增强了部分指令的功能,并进行了一定的扩充。表 5-8 列出了 80286 增加及增强指令的功能。

80386 和 80486 是 32 位的微处理器,具有 32 位的内部通用寄存器和 32 位数据总线,可以进行 32 位数据的并行操作。它们的指令系统中除加强了 80286 部分指令的功能外,主要是增加了对 32 位数的操作,下面主要介绍 80386/80486 新增指令的功能。

表 5-8 80286 新增或增强指令及其功能

指令类别	增强指令	增加指令	新增指令功能说明
数据 传送	PUSH imm	PUSHA POPA	将 8 个通用寄存器的内容按以下顺序压入或弹出堆栈: AX, CX, DX, BX, SP, BP, SI, DI。其中, SP 为执行该指令前的 SP 值
算术 运算	IMUL OPRD1, OPRD2, imm		有符号数乘法。OPRD2 \times imm 结果送 OPRD1。imm 可以是 8 位/16 位常数。乘积超出 16 位时, 高位部分被丢掉
	IMUL reg, imm		有符号数乘法。reg \times imm 结果送 reg。其他与上条指令同
移位 指令	指令码 OPRD, imm		对 8 条循环及非循环移位指令, 移位次数可由立即数 (1 ~ 31) 直接给出
串 操作		[REP] INS [ES: DI], DX REP INSB REP INSW	串输入指令, 从 DX 指定的端口地址中输入一个字节或字送 ES: DI 寻址的存储区, 并自动修改地址指针, 以实现一串数据的输入
		[REP] OUTS DX, [DS: SI] REP OUTSB REP OUTSW	串输出指令, 将 DS: SI 寻址的存储单元中的字节或字送 DX 指定端口中, 并自动修改地址指针, 以实现一串数据的输出。
高级 语言类		BOUND reg, mem (数组边界检查指令)	源操作数是两个存储单元, 其内容分别表示上界和下界。指令用于测试目标寄存器中的内容是否属于上下界之内, 若不属于则产生 5 号中断; 满足则不做任何操作
		ENTER OPRD1, OPRD2 (设置堆栈空间指令)	为高级语言正在执行的过程设置堆栈空间。OPRD1 是 16 位常数, 表示堆栈区域的字节数; OPRD2 是 8 位常数, 表示允许过程嵌套的层数。
		LEAVE (撤销堆栈空间指令)	撤销 ENTER 指令所设置的堆栈空间。一般与 ENTER 指令配对使用
控制 保护类	LAR(装入访问权限) LGDT(装入全局描述符表) LIDT(装入 8 字节中断描述符表) LLDT(装入局部描述符表) LTR(装入任务寄存器) LMSW(装入机器状态字) VERR(存储器或寄存器读校验) ARPL(调整已请求特权级别)	LSL(装入段限符) SGDT(存储全局描述符表) SIDT(存储 8 字节中断描述符表) SLDT(存储局部描述符表) STR(存储任务寄存器) SMSW(存储机器状态字) VERW(存储器或寄存器写校验) CLTS(清除任务转移标志)	

注: reg——寄存器操作数, mem——存储器操作数, imm——立即数

1. 数据传送指令

1) 带符号扩展的数据传送

指令格式:

MOVSX reg, reg/mem

这里,源操作数是8位或16位的寄存器或存储器,目标操作数是16位或32位的寄存器。该指令的功能是将源操作数的符号位扩展后送到目标地址。若源操作数是8位,则扩展为16位;源操作数是16位,则扩展为32位。扩展时符号位为0,则高位都扩展成0;符号位为1,则高位就全部是1。

这条指令将8086指令系统中的字位扩展指令与数据传送指令结合在一起,使这两步操作可以连续进行,主要用于初始化寄存器的高位部分。例:

MOV BX,8000H

MOVSX EAX, BX ;使 EAX = FFFF8000H

2) 带零扩展的数据传送

指令格式:

MOVSX reg, reg/mem

与MOVSX有同样的格式和操作,只是将高位全部扩展为0。例:

MOV BX,8000H

MOVSX EAX, BX ;使 EAX = 00008000H

2. 算术运算指令

在80486中新增加了互换并相加指令。指令格式为

XADD reg/mem, reg

指令中的操作数可以是8位、16位或32位的寄存器或存储器,指令的执行将目标操作数和源操作数相加,结果送回目标地址;同时,目标地址中的原值送入源操作数地址中。同加法指令一样,指令的执行会对状态标志位产生影响。例:

MOV AX,1122H

MOV BX,3344H

XADD AX, BX

指令执行后, $AX = 1122H + 3344H = 4466H$, $BX = 1122H$ 。

3. 逻辑运算和移位指令

在这类指令中新增加了三操作数的双精度的左移和右移指令。

1) 双精度左移指令

指令格式:


```
SHLD reg/mem, reg, imm8
SHLD reg/mem, reg, CL
```

指令中的第一个操作数是 16 位或 32 位的目标地址,存放移位后的结果。第二个操作数是待移动的数据,第三个操作数是移动的位数,可以是 8 位立即数或由 CL 指定。例:

```
SHLD AX, BX, 4 ;将 BX 内容左移 4 位,结果送 AX
```

2) 双精度右移指令
指令格式:

```
SHRD reg/mem, reg, imm8
SHRD reg/mem, reg, CL
```

指令的格式与 SHLD 类似,只是执行的是右移的操作。

4. 位操作指令

1) 位测试与置位指令

这类指令的格式及其功能见表 5-9。

表 5-9 位测试与置位指令

指令格式	功 能
BT reg/mem, reg BT reg/mem, imm	测试目标操作数中由源操作数所指定的位的状态,并将该位的状态复制到进位标志位 CF 中
BTC reg/mem, reg BTC reg/mem, imm	测试目标操作数中由源操作数所指定位的状态,并将该位取反后复制到 CF 中
BTR reg/mem, reg BTR reg/mem, imm	测试目标操作数中由源操作数所指定位的状态,并在将该位复制到 CF 中后将该位清“0”
BTS reg/mem, reg BTS reg/mem, imm	测试目标操作数中由源操作数所指定位的状态,并在将该位复制到 CF 中后将该位置“1”

例:

```
MOV CX, 4
BT [SI], CX ;测试 SI 内容所指向的存储器单元数据的第 4 位,并将该位
;的状态复制到 CF 中
```

2) 位扫描指令
指令格式:

```
BSF reg, reg/mem
BSR reg, reg/mem
```

BSF 指令将源操作数从低位到高位扫描测试,在遇到第 1 个 1 时,将其所在的位号送目标地址 reg 中。若源操作数的所有位均为 0,则 ZF 置 1,否则 ZF = 0。

BSR 的功能与 BSF 完全相同,只是扫描方向是从高位到低位。

5. 根据标志位的状态将字节数量 1 的指令

这类指令的一般格式为

SETcc,OPRD

cc 表示 FLAGS 中标志位的状态,OPRD 是 8 位的寄存器或存储器一个单元。指令的操作是首先判断 cc 给出的条件是否满足,满足则使 OPRD = 1;否则 OPRD = 0。例:

SETNZ AL ;若 ZF = 0,则 AL = 1;否则 AL = 0

SETNS BYTE PTR[SI] ;若 SF = 0,则[DI] = 1;否则[DI] = 0

6. Cache 管理指令

80486CPU 支持二级高速缓冲存储器(Cache)。其中一级缓存在 CPU 内部,这类指令就是用来管理 CPU 内部 8 KB Cache 的,它们共有 3 条,其格式和功能分别为:

1) 作废 Cache 指令 INVD

该指令用来告诉 CPU 一级高速缓存内部的数据失效。执行时先清空内部 Cache,并产生一个专门的总线周期来刷新外部的二级 Cache。指令在执行时将禁止外部 Cache 中的数据写回到主存。

2) 写回和作废指令 WBINVD

指令先刷新内部 Cache,再分配一个专用的总线周期将外部 Cache 的内容写回主存,并在之后的一个总线周期将外部 Cache 刷新。

3) 作废 TLB 项指令 INVLPG

由第 3 章已知,80386 的内部包括一个负责将线性地址转换成物理地址的分页部件,页描述符高速缓冲存储器(TLB)可加速地址的转换。

INVLPG 指令用于使 LTB 中的某一项作废。如果 LTB 项中含有一个存储器操作数映像的有效项,则该 LTB 项被标记为无效。

这 3 条 Cache 管理指令均为零操作数指令。

除以上新增指令外,80386/80486 还扩展了串操作指令的功能,使其能够实现对 32 位数的操作。

5.4.3 汇编语言源程序结构

1. 汇编语言及汇编程序

计算机语言的发展经历了由机器语言到汇编语言再到高级语言这样一个过程。机器语

言和汇编语言都是面向具体机器的语言,与具体的计算机、或更确切地讲与具体的 CPU 有关,用这两种语言编写的程序只能在“适合”于它们的计算机上运行,即程序中所使用的指令必须属于运行该程序的处理器指令系统。而高级语言程序则不存在这个问题,它们多数只与操作系统有关,而与机器硬件系统无关。所以,一般又将机器语言和汇编语言通称为低级语言。它们在当前高级语言功能越来越强大的情况下依然存在、特别是汇编语言还得到广泛地应用,是因为它们本身所具有的高级语言不可代替的优势。

机器语言(Machine Language)是用二进制数码来表示指令和数据的语言。它是计算机惟一能够直接理解和执行的语言,具有执行速度快,占用内存少等优点。用任何语言编写的程序,最终都须通过编译系统翻译成机器代码才能运行。但机器语言不直观,不易理解和记忆,因此编写、阅读和修改程序都比较麻烦。

汇编语言(Assembly Language)弥补了机器语言的不足,它用指令的助记符、符号地址、标号和伪指令等来书写程序。由于助记符接近于自然语言,因此与机器语言相比,它在程序的编写、阅读和修改方面都比较方便,不易出错,而执行速度和机器语言程序近似。另外,它还能够直接对机器硬件进行操作(如直接对内存或端口等)。因此,汇编语言目前在实时控制系统、系统底层功能设计(如机器自检、系统初始化)等方面仍得到广泛地应用。

使用汇编语言编写程序时,要求对它适用的计算机的内部结构和原理有一定的了解。这是因为汇编语言是面向具体机器的语言,不同种类的 CPU 具有不同的汇编语言,互相之间不能通用(但同一系列的 CPU 是向前兼容的)。例如 X86 系列 CPU(包括 Intel 公司的 8088/8086/.../Pentium 和 AMD 公司的 K5/K6/K7 等)的汇编语言程序就不能在 PowerPC 系列的 CPU 上运行。这也是它与高级语言很本质的区别之一。

用汇编语言编写的程序称为汇编语言源程序(源程序文件须有扩展名 .ASM)。与高级语言程序类似,要使一个用汇编语言编写的程序在机器上运行,也必须经过这样几个过程:

① 汇编 即对源程序的编译。由于计算机只能辨认和执行机器语言,因此,必须将汇编语言源程序“翻译”成能够在计算机上执行的机器语言(称为目标代码程序,文件扩展名为 .OBJ),这个翻译的过程称为汇编(Assemble),完成汇编过程的系统程序叫做汇编程序(Assembler)。常用的汇编程序叫做宏汇编(MASM),它功能较强,除了能将源程序翻译成目标代码外,还提供了很多增强功能。如允许使用宏定义以简化编程,能检查出源程序编写过程中出现的语法错误,如非法标号、未定义的助记符等。另外,还可根据用户要求,自动分配各类存储区(程序区、数据区等),自动将非二进制数转换为二进制数,自动进行字符到 ASCII 码的转换以及计算指令中表达式的值等。

② 链接 链接的过程实际上是一个程序重定位的过程。它将各程序模块链接在一起,存放于内存的一个连续区域中,并生成可执行文件 .EXE。

③ 调试 经过以上两个步骤之后,程序就可以在机器上运行了。虽然在汇编过程中已检查出、并改正了源程序中出现的语法错误,但程序中的逻辑错误编译系统是检查不出来的,程序功能的正确和完善还需要程序员通过调试实现。目前常用的汇编语言调试环境为 Turbo Debug(TD),它具有通常的“断点设置”、“单步执行”等调试手段,能同时显示出程序区、数据区、寄存器区、标志位及堆栈区的变化,是比较理想的调试环境。

2. 汇编语言源程序的结构

一个完整的汇编语言源程序通常由若干个逻辑段(Segment)组成,包括数据段、附加段、堆栈段和代码段,它们分别映射到存储器中的物理段上。每个逻辑段以 SEGMENT 语句开始,以 ENDS 语句结束,整个源程序用 END 语句结尾。这些语句称为“伪指令”。伪指令是 CPU 不执行的指令,而由汇编程序识别。主要用于分配内存、定义变量等。常用的部分伪指令说明请参见附录 D。

源程序中所有的指令码都放在代码段中,而数据、变量等则放在数据段或附加段中。程序中可以定义堆栈段,也可以不定义,而利用系统中的堆栈段。具体一个源程序中要定义多少个段,要根据实际需要来定。但一般来说,一个源程序中可以有多个代码段,也可以有多个数据段、附加段及堆栈段,但在同一个程序模块中,只允许有一个数据段、一个附加段、一个堆栈段和一个代码段。

将源程序以分段形式组织,是为了在程序汇编后,能将指令码和数据分别装入存储器的相应物理段中。下面是一个完整的汇编语言源程序的结构。

```
段名 1 SEGMENT
:
段名 1 ENDS
段名 2 SEGMENT
:
段名 2 ENDS
:
段名 n SEGMENT
:
段名 n ENDS
END
```

编写源程序时必须严格按照源程序的结构。在定义完各逻辑段后,需用伪指令 ASSUME(设定段寄存器伪指令)说明定义的段名是什么段(即段的性质),另外,还要对除代码段之外的各个段寄存器初始化。下面通过一个具体的例子来说明一个完整的汇编语言程序的结构。

例 5-27 编写一个两个字相加的程序。程序如下：

```

DSEG      SEGMENT                ;定义数据段
DATA1     DW 0F865H              ;定义被加数
DATA2     DW 360CH              ;定义加数
DSEG      ENDS                  ;数据段结束
;
ESEG      SEGMENT                ;定义附加段
SUM        DW 2 DUP(?)           ;定义 4B 存放结果区
ESEG      ENDS                  ;附加段结束
;
SSEG      SEGMENT                ;定义堆栈段
           DB 200 DUP(?)         ;定义 200B 的堆栈区
SSEG      ENDS                  ;堆栈段结束
CSEG      SEGMENT                ;定义代码段
ASSUME     CS:CSEG,DS:DSEG,
           ES:ESEG,SS:SSEG      ;说明程序中定义的各段的属性
START:     MOV AX,DSEG
           MOV DS,AX             ;初始化 DS
           MOV AX,ESEG
           MOV ES,AX            ;初始化 ES
           MOV AX,SSEG
           MOV SS,AX
           LEA SI,SUM           ;存放结果的偏移地址送 SI
           MOV AX,DATA1         ;取被加数
           ADD AX,DATA2         ;两数相加
           MOV ES:[SI],AX       ;和送附加段的 SUM 单元中
           HLT
CSEG      ENDS                  ;代码段结束
           END START            ;源程序结束

```

本例中没有用到堆栈操作,也可以不用定义堆栈段。各逻辑段的段名可以任起,但不能与指令助记符及伪指令等保留字同名。

习 题 五

5.1 什么叫寻址方式? 8086CPU 共有哪几种寻址方式? 与此相比, 80X86CPU 新增了哪几种寻址方式?

5.2 假设 $(DS) = 212AH$, $(CS) = 0200H$, $(IP) = 1200H$, $(BX) = 0500H$, 位移量 $DATA = 40H(217A0H) = 2300H$, $(217E0H) = 0400H$, $(217E2H) = 9000H$ 。则执行 `JMP WORD PTR[BX]` 指令后, 程序的转移地址为(), 而执行 `JMP DWORD PTR[BX + DATA]` 后, 程序又转向()。

5.3 设堆栈指针 SP 的初值为 $2300H$, $(AX) = 0ABCDH$, $(BX) = 1234H$ 。执行指令 `PUSH AX` 后, $(SP) = ?$, 再执行指令 `PUSH BX` 及 `POP AX` 之后, $(SP) = ?$ $(AX) = ?$ $(BX) = ?$

5.4 指出下列指令的错误:

- | | |
|--|---------------------------------------|
| (1) <code>MOV AH, CX</code> | (2) <code>MOV 33H, AL</code> |
| (3) <code>MOV AX, DS: [SI][BP]</code> | (4) <code>MOV [BX], [SI]</code> |
| (5) <code>ADD BYTE PTR[BP], 256</code> | (6) <code>MOV DATA[SI], ES: AX</code> |
| (7) <code>JMP BYTE PTR[BX]</code> | (8) <code>OUT 230H, AX</code> |
| (9) <code>MOV DS, BP</code> | (10) <code>MUL 39H</code> |

5.5 已知 $(AL) = 7BH$, $(BL) = 38H$, 试问执行指令 `ADD AL, BL` 后, AF 、 CF 、 OF 、 PF 、 SF 和 ZF 的值各为多少?

5.6 试比较无条件转移指令、条件转移指令、调用指令和中断指令有什么异同。

5.7 要将寄存器 BX 的内容与存储器单元 $[1200H]$ 中的内容互换, 可用哪些指令实现?

5.8 试判断下列程序执行后, BX 中的内容:

```
MOV CL, 3
MOV DX, 0F7H
ROL DX, 1
ROR DX, CL
```

5.9 从某外设端口输入一个字节数, 当输入数的 $D1$ 位为 0 时, 程序转向 `NEXT1`, 否则程序转向 `NEXT2`。试编写实现该功能的程序段。

5.10 乘法和除法指令的结果可以放在哪些寄存器中?

5.11 设 $(SP) = 0200H$, $(IP) = 1200H$, 在响应一条 2 字节的 `INT` 指令后, $(SP) = ?$ SP 所指向单元的内容是多少?

5.12 将 $+46$ 和 -38 分别乘以 2, 可应用什么指令来完成? 如果除以 2 呢?

5.13 串操作指令的操作方向由哪个标志位决定? 如果要按减地址方向操作, 应将该标志位置成什么状态?

5.14 已知 $AX = 8060H$, $DX = 03F8H$, 端口 `PORT1` 的地址是 $48H$, 内容为 $40H$; `PORT2` 的地址是 $84H$, 内容为 $85H$ 。请指出下列指令执行后的结果:

- (1) OUT DX, AL
- (2) IN AL, PORT1
- (3) OUT DX, AX
- (4) IN AX, 48H
- (5) OUT PORT2, AX

5.15 以下两条指令完成了什么操作?

```
LEA BX, BLOCK
MOV AX, [BX + 10]
```

5.16 试判断下列指令执行后, AX = ? BX = ? CX = ? DX = ?

```
MOV AX, 10H
MOV BX, 20H
MOV CX, 04H
MOV DX, 03H

SSSS: INC AX
      ADD BX, BX
      SHR DX, 1
      LOOP SSSS
      HLT
```

5.17 80386/80486CPU 主要增强了哪方面的指令?

5.18 汇编语言源程序具有什么样的结构? 在一个程序模块中最多可以定义几个逻辑段?

5.19 试写出一个完整的汇编语言源程序的结构。

第6章 存储系统

本章从存储系统的角度出发,介绍现代微型计算机的三级存储系统结构、存储器的基本工作原理、典型半导体存储器芯片与 CPU 的连接和使用、存储器扩展技术、现代存储器管理技术、外存储器的工作原理和应用以及现代存储器件的技术特点等。

6.1 概 述

计算机基本工作原理是“存储程序控制”。从程序员的角度来看,计算机要开始工作,必须把相应的程序和数据装入存储器才能开始运行。

由前面的章节知道,在程序执行过程中,中央处理器从存储器中取得指令。运算指令中需要的数据也要通过访问存储器指令从存储器中取得。而运算结果在程序结束前必须全部写入存储器中。各种输入/输出设备也直接与存储器交换数据。因此,存储器是计算机运行过程中的信息存储交换的中心设备,从这个意义上说,现代计算机系统是以存储器为中心的。

内存是内部存储器的简称,它是可通过指令中的地址码直接访问的存储器,用来存储正在被 CPU 使用的程序和数据,又叫做主存储器。

外存是外存储器的简称,一般不直接被 CPU 访问,通常存放当前不处于活动状态的程序和数据。外存是主存储器的补充,所以又叫做辅助存储器。

速度、容量是表示一个存储器性能的两个主要指标。速度用存取周期、读出时间、频带宽度等表示,容量用 B、KB、MB、GB 等单位表示。

6.1.1 存储系统概念

存储系统与存储器是两个不同的概念。在现代计算机中通常有多种用途的存储器,如内存、Cache、通用寄存器、磁盘、各种缓冲存储器、磁带、光盘等。它们的构成材料、工作方式也各不相同,但并不能说因此构成存储系统。存储系统的定义是:由两个或两个以上速度、容量和价格各不相同的存储器用软件、硬件或软硬件相结合的方法连接起来成为一个系统。这个系统从程序员的角度看,它是一个存储器整体,这个存储器的速度接近于其中速度最快的那个存储器,存储容量与存储容量最大的那个存储器相等或接近,单位容量的价格接近最便宜的那个存储器。对于一个计算机系统,存储系统的优劣,特别是它的存取速度和存

储容量关系着整个计算机系统的优劣。

现代微型计算机通常具有两种存储系统,一种是由 Cache 和主存储器构成的 Cache 存储系统,如图 6.1 所示。另一种是由主存储器和磁盘构成的虚拟存储系统,如图 6.2 所示。这两种存储系统的作用是不相同的,前者的主要目标是提高存储器的速度;而虚拟存储系统的主要目标是增加存储器的容量。

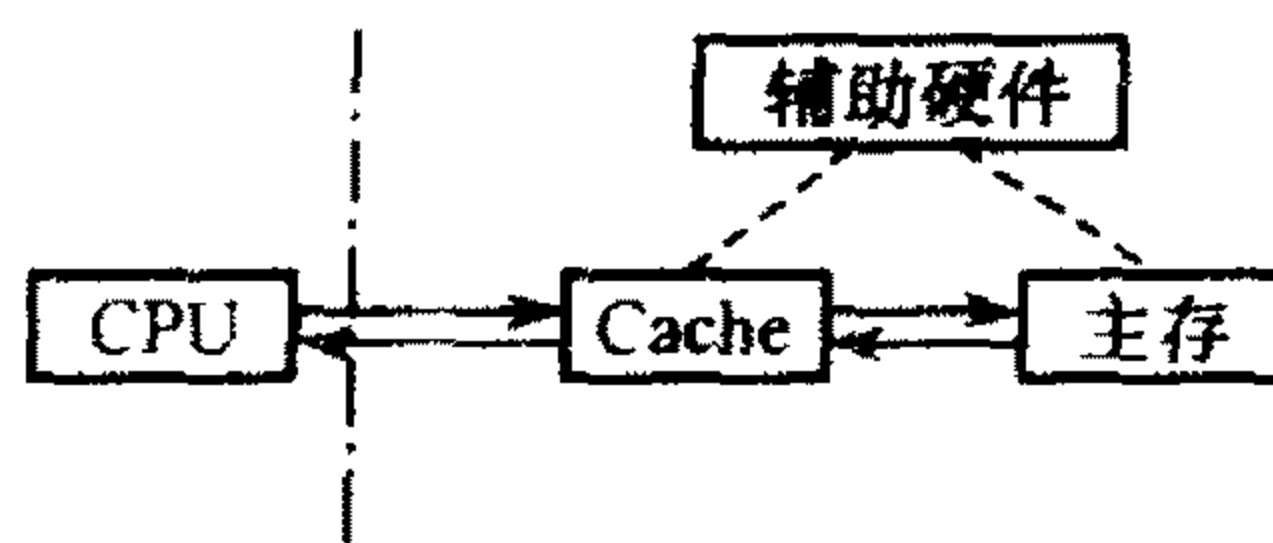


图 6.1 Cache 存储系统

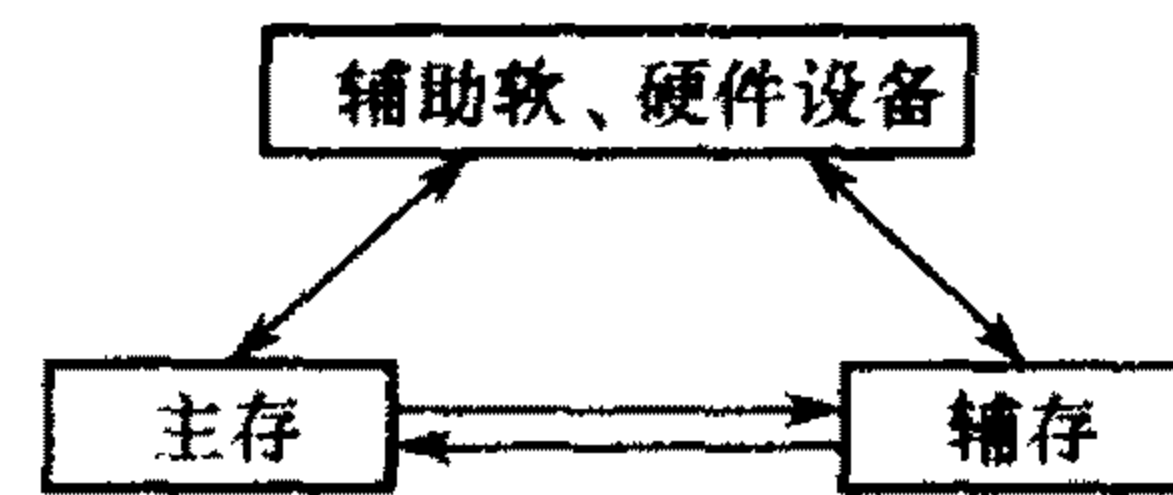


图 6.2 虚拟存储系统

Cache 存储系统的管理全部由硬件实现,无需系统程序员干预,即它对软件开发设计人员是透明的(一个实际存在的部件看起来好像不存在,称为“透明”)。

目前,Cache 一般用高速静态存储器(SRAM)组成,存取周期为十几个乃至几个纳秒,存储容量在几兆字节至几十兆字节之间,价格较高。主存一般由动态存储器(DRAM)组成,存储周期为几十纳秒,存储容量一般在几百兆字节甚至吉字节,价格比 Cache 相对便宜得多。这两种存储器组成存储系统,在一定的程序执行时间段内,CPU 需要的数据都能在 Cache 中访问到,因此,这个存储系统的存取周期与 Cache 非常接近。对于程序员来说,由于 Cache 采用相联式访问,只需要对主存储器编址访问而“看不到”Cache (即不对它进行编址),因此,存储系统的容量就是主存储器的容量。在整个存储系统中,Cache 所占的比例很小,因此,每单位的平均价格与主存储器很接近。

虚拟存储系统由主存储器与外存储器(一般为磁盘存储器)构成。在早期的微型计算机中,磁盘等外存储器作为外部设备的一部分,仅用于长期保存信息。由于内存容量很小,程序员必须花费很大精力把大程序预先分成块,确定好这些程序块在外存设备中的位置和装入主存的地址。并且在运行中还要预先安排好各块如何和何时调入/调出。

现代虚拟存储系统在操作系统的支持下,将主存和外存看作一个整体,用软硬件相结合的方法进行管理。使得程序员能够对主存、辅存统一编址,这就形成一个很大的地址空间,称为虚拟地址空间,比实际主存储器的存储容量大得多,32 位微型计算机可访问的编址空间为 4GB。虚拟存储系统的访问速度接近主存储器。由于磁盘存储器每位的价格比主存储器便宜的多,因此,整个存储系统的每位平均价格接近磁盘存储器。

对存储系统进行编址的要求,除了应对计算机的使用者提供尽可能大的地址空间以外,还应能对这个地址空间进行随机访问。对于虚拟存储系统,由于磁盘存储器实际上不同于主存储器可以随机访问,它的地址空间不能直接被一般指令访问,而主存储器的地址空间对

于使用较大规模程序的用户来说又太小。因此,虚拟地址空间既不是主存储器的地址空间,也不是磁盘存储器的地址空间,它是为使用者另外设计的。这个虚拟地址空间远大于主存储器的实际地址空间,而且采用与主存储器同样的随机访问方式。

要构成一个合理优化的存储系统,仅有上述概念性的目标是不够的,必须具有明确的量化指标,设计者才能有可遵循的技术路线。表示存储系统的性能有3个主要参数有:容量 S 、速度 T 和价格 C ,组成存储系统的每个存储器也有3个同样的参数,通过分析这些参数之间的关系,可以评测一个存储系统。

1. 存储容量 S

如果有两种存储器 M_1 和 M_2 ,它们组成一个存储系统。两种存储器的容量、速度和价格分别为 S_1, T_1, C_1 和 S_2, T_2, C_2 ,存储系统的容量、速度和价格分别为 S, T, C 。

对于 Cache 存储系统,由系统程序员看来,存储系统的容量接近主存储器的容量,故选择主存 M_2 进行编址,对 Cache 在内部采用相联访问方式管理。因此,系统程序员看到的是主存储器的地址空间,存储系统的容量就是主存储器的容量, $S = S_2$ 。

对于虚拟存储系统,它的地址空间比主存储器大得多。还应当说明的是,在一般计算机系统中,并不是整个磁盘存储器都作为虚拟存储系统使用。磁盘存储器的主要用途仍然是用来存放系统软件、应用软件和用户文件的,只有在多任务多用户操作系统的交换区或交换文件才是用来做虚拟存储系统。

2. 存取周期 T (访问周期、存取时间等)

存储系统的存取时间与命中率 H (从 Cache 中访问到数据的概率)关系很大。 H 一般用模拟试验的方法得到。在一组有代表性的程序执行过程中分别统计对 M_1 的访问次数 N_1 和对 M_2 的访问次数 N_2 ,然后代入关系式(6.1):

$$H = \frac{N_1}{N_1 + N_2} \quad (6.1)$$

整个存储系统的存取周期可以用 M_1 和 M_2 两个存储器的存取时间 T_1, T_2 和命中率 H 来表示:

$$T = H \cdot T_1 + (1 - H) \cdot T_2 \quad (6.2)$$

当命中率 $H \rightarrow 1$ 时, $T \rightarrow T_1$,即存储系统的速度接近于较快的 M_1 存储器的存取周期 T_1 。

设存储系统的访问效率为:

$$e = \frac{T_1}{T} \quad (6.3)$$

存储系统的速度与相对较快的那个存储器的速度越接近,访问效率就越高。把式(6.2)代入式(6.3),得到

$$e = \frac{T_1}{H \times T_1 + (1 - H) \times T_2} = \frac{1}{H + (1 - H) \times \frac{T_2}{T_1}} = f\left(H, \frac{T_2}{T_1}\right)$$

可以看出,存储系统的访问效率主要与命中率和构成存储系统的两级存储器的速度之比有关。因此,要提高存储系统的访问效率有两条途径,一是提高命中率 H ,另一条是使构成存储系统的两个存储器的速度之比不要太大。

对于虚拟存储系统,由于磁盘的存取操作还要依赖机械运动,两级存储器的速度相差悬殊,主存储器的存取速度为纳秒级,硬盘存取速度为毫秒级。若要使访问效率 e 比较高(如 $e = 0.9$),需要极高的命中率 H ,如果 $T_2/T_1 > 10^5$,则依上式计算 H 约为 0.999 999,如何使用现有技术达到高命中率呢?

因为磁盘在物理上是以块为单位(每块 512 B)访问的,所以,虽然磁盘存储器的寻址定位时间很长,但当磁盘找到要访问的连续的数据块之后,数据的传输速率还是相当高的。因此,当不命中时,通过操作系统的功能调用,把将要使用的一大批程序和数据都调入主存储器,使得在以后的多次对虚拟存储系统的访问都能在主存储器命中。只要主存储器的容量比较大,能够一次装入比较多的程序和数据,这样,尽管两级存储器的速度差异悬殊,一次不命中需要花费较长的时间进行调度,然而,由于命中率特别高,整个虚拟存储系统的访问效率还是很高的。

Cache 存储系统要缓冲 CPU 和主存之间的速度差异,目前 CPU 与主存储器的速度相差两个数量级,如果要求 $H \approx 0.999$,用一级 Cache 是做不到的。通常采用两级或三级 Cache,再加上 CPU 内部的一些缓冲存储器,像通用寄存器等来提高数据的重复利用率,使得每两级之间的速度比减小。另外,再采用预取技术以大幅度提高命中率:当不命中时,在数据从主存储器取出送往 CPU 的同时,把主存储器相邻几个单元中的数据(一个数据块)都取出来送入 Cache 中。CPU 以后再对 Cache 存储系统进行访问时,命中率就会提高。详细介绍参见 6.5 节。

3. 单位容量的平均价格 C

整个存储系统的单位容量平均价格可以用下式计算:

$$C = \frac{C_1 \times S_1 + C_2 \times S_2}{S_1 + S_2} \quad (6.4)$$

当 S_2 大大超过 S_1 时, $C \approx C_2$ 。这时,整个存储系统的单位容量价格 C 接近于比较便宜的 M_2 存储器的单位容量价格 C_2 。但是 S_2 和 S_1 的差距应在一个合理的范围内,如果差距太大,存储器要达到较高的性能,调度安排将会很困难。

6.1.2 存储器的体系结构

已经知道在一个计算机系统中,对存储器的容量、速度和价格这3个基本性能指标都有一定的要求。存储容量应保证各种应用的需要;存储器速度应尽量与CPU的速度相匹配;存储器的价格应比较合理。然而,这三者经常是互相矛盾的。存储器的速度越快,则单位容量价格越高;存储器容量越大,则速度就越慢。显然,仅仅采用一种技术组成单一的存储器不可能同时满足这些要求。只有采用由多级存储器组成的存储体系,用各种工作速度、存储容量、访问方式、适应不同用途的存储器结合起来,构成一个层次结构,如图6.3所示。图中最内层CPU中的通用寄存器(还包括指令和数据缓冲栈),当运算直接在CPU的通用寄存器中进行时,由于CPU不和主存进行数据交

换,速度最快。但通用寄存器数量有限,如一般的Pentium CPU中有8个32位的通用寄存器。

一级缓存Cache设置在CPU和主存之间,安装在CPU芯片内部,它们的工作速度很高,可以很好地与CPU的速度匹配。二级缓存Cache(有时还有三级Cache)可以放在CPU的内部,也可以放在外部,工作速度逐级降低。若用*i*表示层数,则有

存取周期: $T_i < T_{i+1}$

存储容量: $S_i < S_{i+1}$

由于存放在各级存储器中的程序和数据最终都要送到CPU中进行处理,从CPU的角度看,期望各层次存储器的速度接近中心层次,存储容量和价格接近外部层次,这也正是设计者致力于解决的课题。随着半导体工艺水平的发展和计算机技术的迅速进步,各种存储器的性能指标不断提高,存储器多级结构的构成可能相应地调整,但是,由多级存储结构来构成完整的存储体系这一设计思想是不会改变的。尽管半导体存储器芯片的集成度日趋增高,主存容量不断扩大,但由于系统软件和应用软件的发展,主存容量总是供不应求,只要这一现状存在,由主存—辅存(外存)为主体的多级存储体系也就会长期存在。

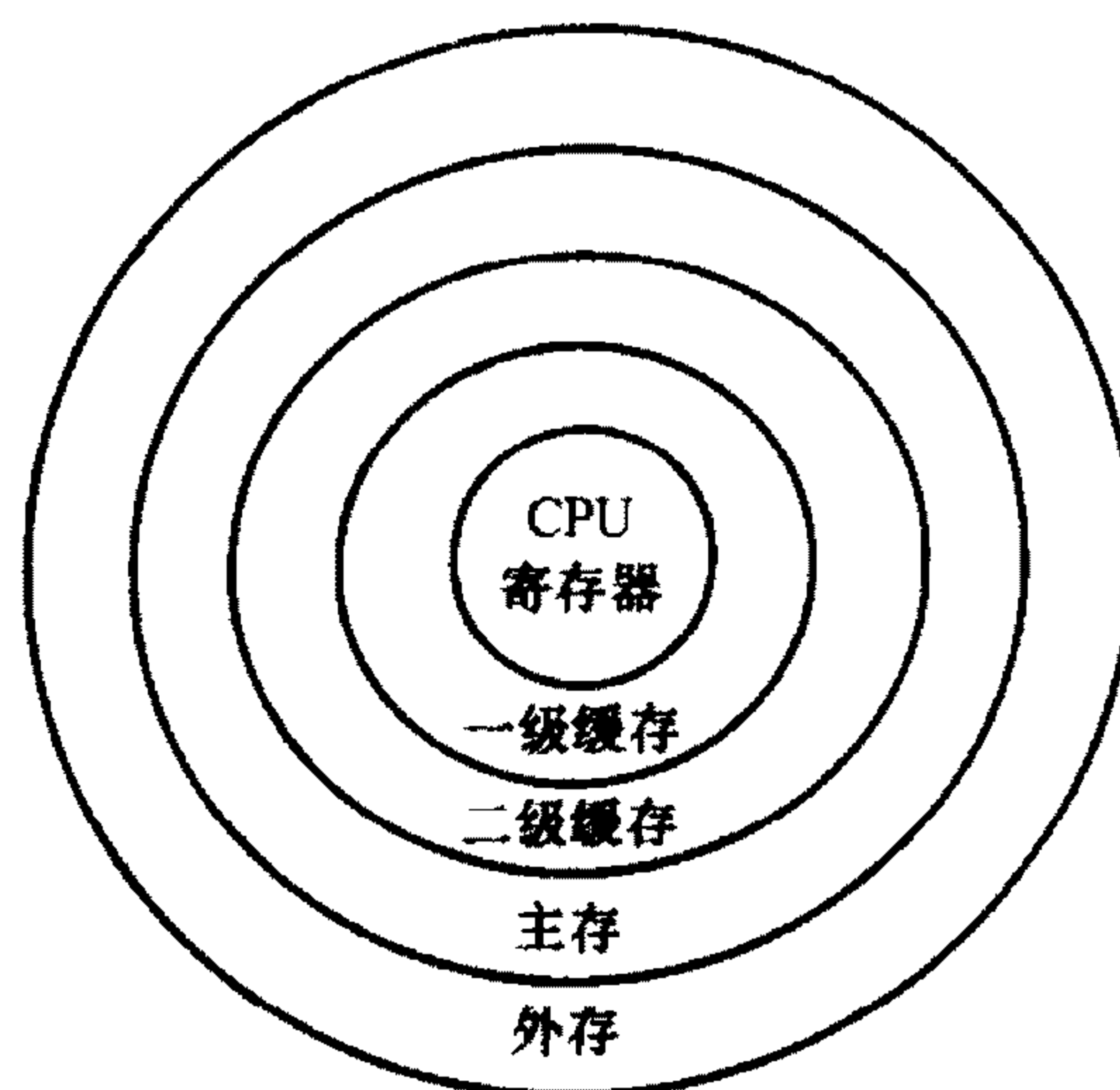


图 6.3 存储器的层次结构

6.1.3 存储器的分类

1) 存储器的种类很多,可根据其介质、用途和寻址方式分类。

存储器按存储介质分类可分为:

① 以磁性材料为存储介质的磁存储器 如磁盘(软盘、硬盘)、磁带、磁卡、磁鼓等。

② 电子介质的半导体存储器 如静态 RAM、动态 RAM、ROM、闪速存储器等。

半导体存储器按照工作方式又可分为:

- 读/写存储器 RAM(也叫随机存储器)
- 只读存储器 ROM

③ 光介质的激光存储器 如广泛应用的只读光盘。

2) 按用途以及与 CPU 的关系分类

① 内存储器(Intenal Memory) 又称主存储器 MM(Main Memory)。目前主要由 CMOS 半导体集成电路组成,用来存放计算机运行期间要执行的程序与数据。

② 控制存储器(Control Memory, CM) 对采用微程序控制的 CPU, CM 是存放控制信息,即存放微程序的存储器,由高速只读存储器 ROM 构成, CM 集成在 CPU 结构内部。

③ 高速缓冲存储器(Cache) 用来存放主存中处于活动的部分(正在执行的程序和正在使用的数据)的副本,以解决主存速度的不足,由 TTL(Transistor - Transistor Logic)、ECL(Emitter Coupled Logic)等高速半导体存储元件构成。

④ 外存储器(External Memory) 用来存放当前不参与运行的大量信息。当需要用这些信息时, CPU 要先调入主存后才能使用,其特点是:比主存容量大,价格低,但速度也慢。磁盘、磁带和光盘是目前常用的外存储器。

3) 按寻址方式分类

① 随机存取存储器(Random Access Memory, RAM) 若能对任意位置的存储单元进行访问,而且读/写时间与存储单元所处位置无关,这种存储器就称“随机存取存储器”。主存储器、控制存储器及高速缓存都是由随机存取存储器构成的。

② 顺序存储器(Serial Access Memory, SAM) 只能顺序读/写存储单元的存储器称为“顺序存储器”。磁带是典型的顺序存储器,只有被访问的单元通过固定的读/写磁头时,才能读/写它,所需要的读/写时间取决于该单元与读/写磁头的相对位置。处于不同的位置,所需的读/写时间是不同的。在 SAM 中,一般只能用平均读/写时间作为使用参数。顺序存储器的优点是结构简单,成本低。

③ 直接存储器(Direct Access Memory, DAM) 磁盘即属此类存储器,它的寻道过程可看作随机方式,而在同一磁道环上是按顺序存取的,所以是介于前两类之间的一类存储器,由于它是对一个小的区域——扇区直接寻址,故称直接存储器。

存储器的分类表如图 6.4 所示。

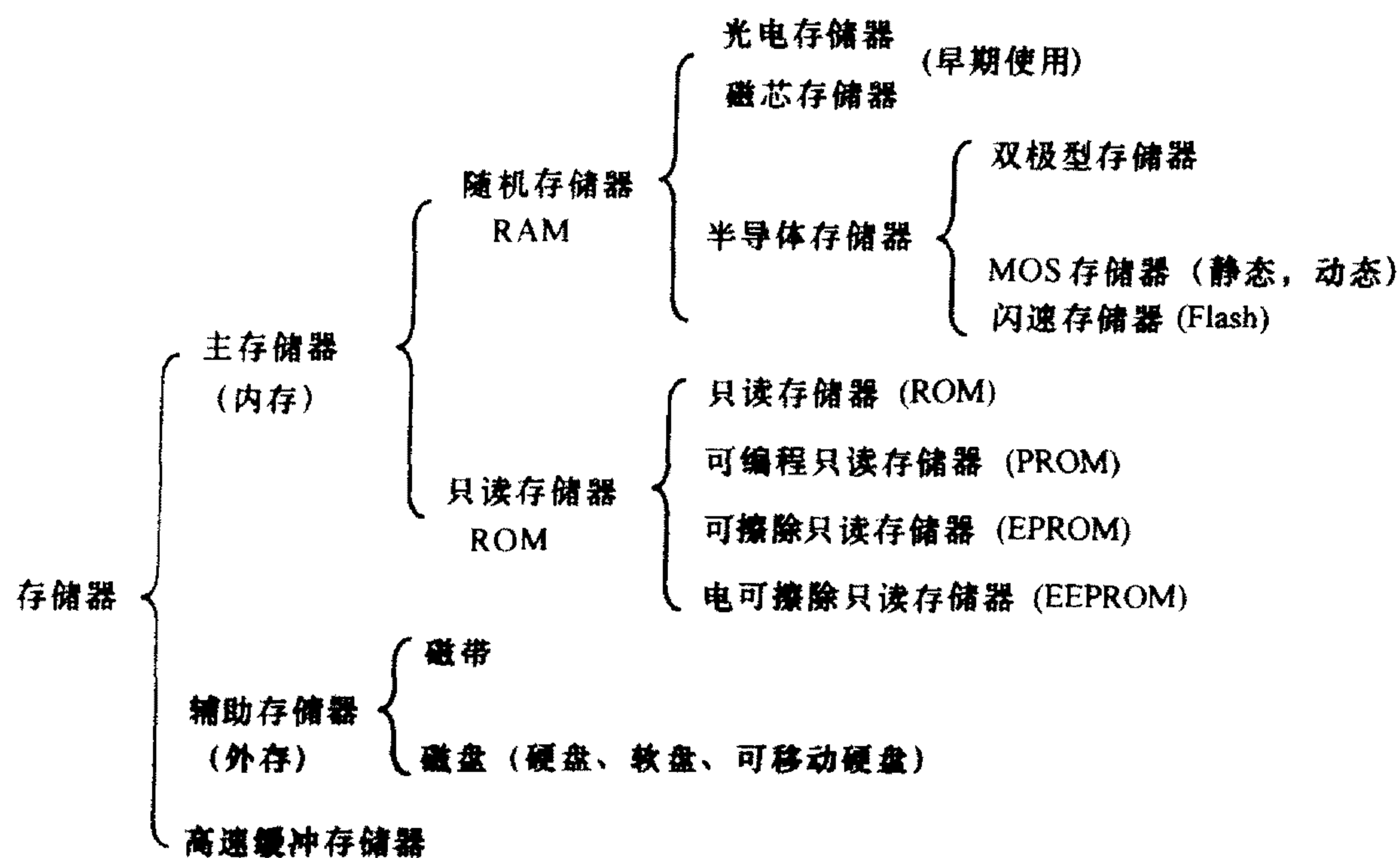


图 6.4 存储器分类

6.1.4 存储器的主要性能指标

1. 存储容量

存储容量是指存储器所能容纳的二进制信息总量,通常以字节表示。常用单位有字节(Byte, B),千字节(Kilo Byte, KB),兆字节(Mega Byte, MB)和吉字节(Giga Byte, GB)为单位, $1\text{ KB} = 2^{10}\text{ B}$, $1\text{ MB} = 2^{20}\text{ B}$, $1\text{ GB} = 2^{30}\text{ B}$ 。存储容量用“存储单元个数 \times 每存储单元的位数”来计算。例如,SRAM 芯片 6264 的容量为 $8\text{ K} \times 8\text{ bit}$,即它有 8 K 个单元($1\text{ K} = 1\text{ 024}$),每个单元存储 8 位(一个字节)二进制数据。各半导体存储器生产厂家提供了许多种不同容量的存储器芯片,用户在构成计算机内存系统时,可以根据要求加以选用。当然,当计算机的内存确定后,选用容量大的芯片则可以少用几片,这样不仅使电路连接简单,而且也可以降低功耗。

2. 存储器速度

① 存取时间 又称存储器访问时间,即启动一次存储器操作(读或写)到完成该操作所需要的时间。CPU 在读/写存储器时,其读/写时间必须大于存储器芯片的额定存取时间。如果不能满足这一点,计算机则无法正常工作。存取时间取决于存储介质的物理特性及读/写机构的特性。

② 存取周期 连续启动两次独立的存储器操作所需间隔的最小时间。若令存取时间为 t_A ,存取周期为 T_C ,则二者的关系为 $T_C \geq t_A$ 。存取周期往往比存取时间要大。如动态

RAM 存储器,在每次读操作时,原存信息被破坏,必须把读出信息重新写入原来单元加以恢复,使得存取周期 T_c 等于读数时间与写入时间之和。

③ 存储器带宽 单位时间内存储器可读/写的字节数(或二进制的位数)称为存储器的带宽,记作 B_m 。带宽除了与存储周期有关外,还与存储器一次可读/写的二进制位数有关。带宽反映的是存储器的数据吞吐速率。常称为存储器的数据传输率。

3. 可靠性

计算机要正确地运行,必然要求存储器系统具有很高的可靠性。内存发生的任何错误会使计算机不能正常工作。而存储器的可靠性直接与构成它的芯片有关。目前所用的半导体存储器芯片的平均故障间隔时间(MTBF)约为 $5 \times 10^6 \sim 1 \times 10^8$ 小时左右。

4. 功耗

使用功耗低的存储器芯片构成存储系统,不仅可以减少对电源容量的要求,而且还可以提高存储系统的可靠性。

6.2 随机存储器(RAM)

随机存取存储器 RAM 主要用来存放当前运行的程序、数据、中间结果及堆栈等,其内容既可随时读出,也可随时写入和修改,掉电后内容将全部丢失。常用的半导体存储器主要有两类:一类是双极型半导体存储器 TTL 和发射极耦合电路存储器 ECL,优点是速度快(TTL 存储芯片速度约为 $10 \sim 25$ ns, ECL 存储芯片速度 ≤ 10 ns,驱动能力强,缺点是集成度低,功耗大,价格高,多用于小容量高速存储器。另一类是金属氧化物场效应晶体管(Metal Oxide Semiconductor, MOS)存储器,又可分为: N 沟道金属氧化物场效应晶体管 NMOS、P 沟道金属氧化物场效应晶体管 PMOS、互补金属氧化物场效应晶体管 CMOS。MOS,优点是集成度高,功耗小,成本低,但速度较低,如 NMOS 存储芯片速度约为 $50 \sim 100$ ns,主要用于大容量存储器。MOS 存储电路又分静态存储器和动态存储器两类。

本节将从应用的角度出发,以几种常用的典型芯片为例,详细介绍两类 MOS 型读/写存储器(SRAM 和 DRAM)的特点、外部特性以及它们的应用。

6.2.1 存储器的一般概念

1) 存储信息的原理

为了存储二进制代码,存储器中要有存储元件。目前大量使用的是半导体存储器、磁存储器和光存储器。下面先介绍存储 1 位二进制的基本存储单元。

存储器由一些能够表示二进制 0 和 1 的状态的物理器件组成,这些器件本身具有记忆功能。在半导体存储器中,存储信息是以晶体管导通与否来表示的。假设双稳态触发电路(图略)中有晶体管 T_0 ,如果使 T_0 导通,有电流流过 T_0 时设为存储“0”;反之, T_0 截止,无电流流过 T_0 (或者 T_1 导通,有电流流过 T_1),即为存储“1”。

在磁存储器中,是用铁磁材料的剩磁方向来表示二进制状态的。当读/写磁头中写入电流的方向不同,磁路中磁通方向就不同。在存储体——磁层上就会留下不同磁化方向的剩磁。表现在磁滞回线上就处于磁矩 $+B_m$ 或 $-B_m$ 。如果把剩磁 $+B_m$ 当作存储 1,则 $-B_m$ 就是存储“0”。

存储介质必须具备以下条件:

- ① 有两个稳定的能量状态,并由一个高能势垒将两个状态分开;
- ② 借助外部能源能使两个状态之间进行无限次转换(即可写);
- ③ 借助外部能源能获知其状态(即可读)。半导体存储器和磁存储器都具备这些条件。

2) 基本存储单位与存储容量计算

存储一位二进制代码的存储元件称为基本存储单元(Bit),每 8 个基本存储单元组成的一个基本单位称为一个字节(Byte),计算机系统对于每个字节编排一个地址,形成一个编址的存储单元。为相区别,上述的最小存储体基本单位叫做一个存储元。多个(一般为 2^n 个)存储单元组成存储体。

存储单元的地址是用二进制编码表示的,相邻单元的地址相差为 1, N 个单元需要的编址二进制码为 n 位,其关系式为

$$N = 2^n$$

存储器中存储单元的总数称为存储容量。显然,存储容量越大,计算机的信息处理能力也就越强。我们已经知道,存储容量的单位为字节(B)、千字节(KB)或兆字节(MB),例如 64 KB、128 MB 等,典型微型计算机 8088/8086 CPU 可以访问的内存容量为 1 MB。

上文提到存储器有两种基本操作——读和写。读操作不破坏存储单元中原有的内容,即读操作是非破坏性的操作。而写操作使得新写入的数据覆盖原有的内容,所以写操作是破坏性的。

6.2.2 静态随机存储器(SRAM)

1. SRAM 的工作原理

静态 RAM 的基本存储电路(即存储元)一般是由 6 个 MOS 管组成的双稳态电路,如图 6.5 所示,其中两个管 T_1 、 T_2 的状态即为存储的信息, T_1 截止, T_2 导通为状态“1”; T_2 截止, T_1 导通为状态“0”。

而负载管 T_3 和 T_1 构成一个反相器,负载管 T_4 和 T_2 构成另一个反相器。这两个反相器彼此交叉反馈,构成一个双稳态触发器。另有两个控制门管 T_5 、 T_6 接 X 地址选择线(又称字线),当 X 地址选择线为高电平时, T_5 和 T_6 导通,使双稳态电路与读/写电路连接,可对其进行写入或读出。当 X 地址选择线为低电平时, T_5 和 T_6 都断开,双稳态电路和读/写电路脱离,依靠自身的交叉反馈保持原状态(即所存信息)不变。

读/写电路通过一对位线: \bar{W} 和 W 与双稳态电路连接,在写操作时,若要写入“0”,则使 W 为低电平, \bar{W} 为高电平,经导通的 T_5 管迫使 A 点的电位降低,则使 T_1 管导通而 T_2 管截止,若要写入“1”,使 W 降为低电平经导通的 T_6 迫使 B 点变为低电平,使得 T_1 截止而 A 点电位升高,从而 T_2 管导通。交叉反馈将加快这种状态变化。当输入信号和地址选择信号消失后, T_5 、 T_6 、 T_7 、 T_8 截止, T_1 、 T_2 就保持被写入的状态。只要不掉电,写入的信息就能保持不变。

读操作时, X 地址选择线加高电平,使门管 T_5 、 T_6 导通,若原存信息为 0,状态为 T_1 管导通而 T_2 管截止。 \bar{W} 通过 T_5 到 T_1 到地形成放电回路,在位线 \bar{W} 产生电压降,有电流经过,经差动放大器检测出“0”信号。若该位原存“1”,则使得位线 \bar{W} 产生电压降,经放大检测为“1”信号。总之,读出时根据位线有无电流判明存储信息, \bar{W} 上有电流为“0”, W 上有电流为“1”。

读出过程中,位线变成了读出线。读取过程不影响触发器原来的状态,称为非破坏性读出。

一般字线分为 X 地址选择线和 Y 地址选择线,即存储电路有两个地址输入端,是为了能进行双重译码选择。若 T_7 、 T_8 为 Y 向公用选择控制管,当 Y 选择线为高,选中一行, X 选择线为高,选中一行, X 、 Y 同时为高,其交点上的存储元才被最后选中。双向译码的好处是:对于大容量存储芯片,由于其集成密度高,地址码的位数将大大增加,将地址码分来两次输入,芯片的地址引线只需地址码位数的一半,可以大幅度降低地址译码器的复杂程度。

SRAM 在微型计算机领域有着极其广泛的应用。下面就以结构较简单的 SRAM 芯片 6264 为例,说明它的基本外部特性及工作过程。

2. 6264 存储芯片的引脚及其功能

6264 芯片是一个 $8\text{ K} \times 8\text{ bit}$ 的 CMOS SRAM 芯片,其引脚如图 6.6 所示。它共有 28 根引出线,包括 13 根地址线、8 根数据线以及 4 根控制信号线,它们分别定义为:

- 地址信号线 13 位: $A_0 \sim A_{12}$, 13 根地址信号线上的地址编码最大为 2^{13} , 即 8 192(8K)

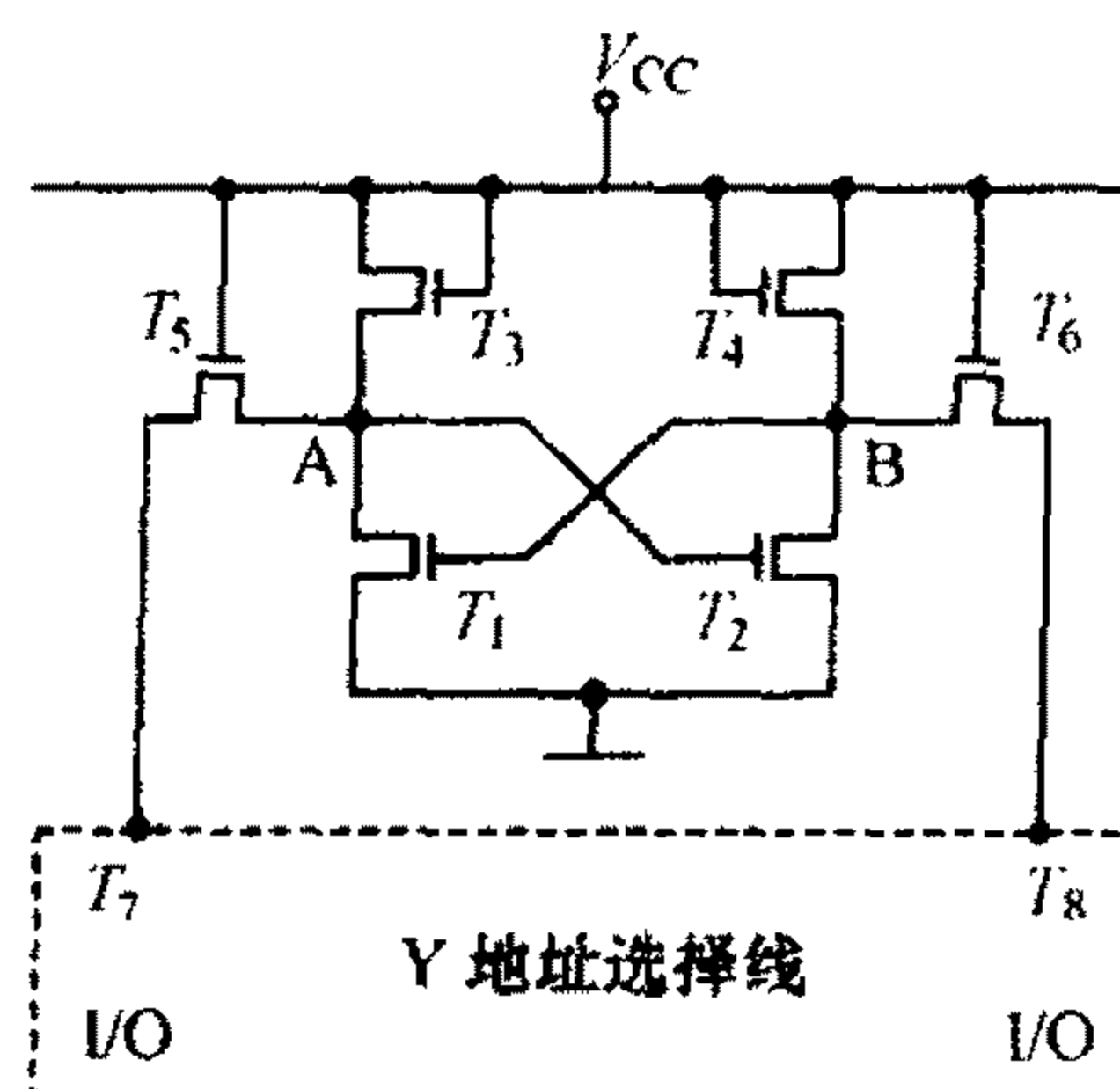


图 6.5 静态 RAM 的基本存储电路

个。也就是说 6264 的 13 根地址线上的二进制状态信号经过内部译码,可以选中芯片本身 8K 个存储单元的其中之一。这 13 根地址线通常连接到系统地址总线的低 13 位上,以便 CPU 能够寻址芯片上的各个单元。

- 双向数据线 8 位: $D_0 \sim D_7$, 对应于片内各编址单元并行读/写的 8 位。即片中每个单元包含 8 个存储元,构成一个字节。使用时,这 8 根数据线与系统的数据总线相连,与 CPU 进行数据信号传送。

- 片选信号线 \overline{CS}_1 、 CS_2 , 当 \overline{CS}_1 为低电平、 CS_2 为高电平 ($\overline{CS}_1 = 0, CS_2 = 1$) 时,该芯片被选中,CPU 才可以对它进行读/写。事实上,一个微型计算机系统的内存空间是由若干块存储器芯片组成的,某块芯片映射到内存空间的哪一个位置(即处于哪一个地址范围)上,是由高位地址信号决定的。系统的高位地址信号和控制信号通过译码产生选片信号,将芯片映射到所需要的地址范围上。早期的 8086/8088 CPU 有 20 根地址线,若连接 SRAM6264,系统的高位地址信号就是 $A_{13} \sim A_{19}$ 。有关地址译码,将在下面详细介绍。

- \overline{OE} : 输出允许信号。只有当 \overline{OE} 为低电平时,CPU 才能够从芯片中读出数据。

- \overline{WE} : 写允许信号。当 \overline{WE} 为低电平时,允许数据写入芯片;而当 $\overline{WE} = 1, \overline{OE} = 0$ 时,允许数据从该芯片读出。

表 6-1 总结了以上 4 个控制信号的功能。

表 6-1 6264 真值表

\overline{WE}	\overline{CS}_1	CS_2	\overline{OE}	$D_0 \sim D_7$
0	0	1	x	写入
1	0	1	0	读出
x	0	0	x	三态 (高阻)
x	1	1	x	
x	1	0	x	

- 其他引线: V_{CC} 为 +5 V 电源,GND 是接地端,NC 表示空端。

3. 6264 的工作过程

要使芯片正常工作,必须依照额定的时序关系提供地址、数据信息和有关控制信号。

6264 写入数据的过程是: 首先把要写入单元的地址送到地址线 $A_0 \sim A_{12}$ 上;要写入的数

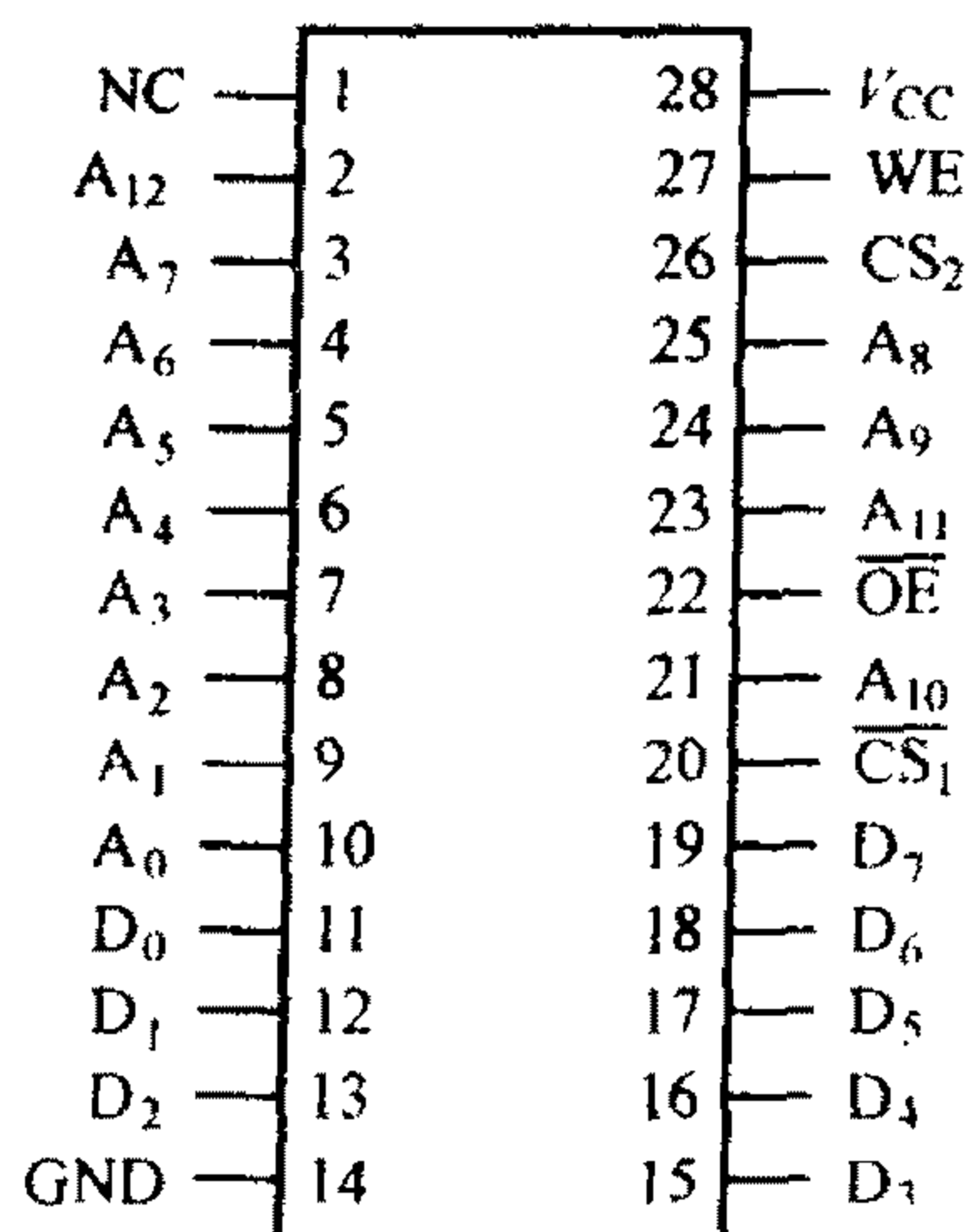


图 6.6 SRAM 6264 外部引脚图

据送到数据线上;然后使 \overline{CS}_1 、 CS_2 同时有效($\overline{CS}_1 = 0$, $CS_2 = 1$);再在 \overline{WE} 端加上有效的低电平, \overline{OE} 端状态可以任意。这样,数据就可以写入指定的存储单元中。写入过程的工作时序如图 6.7 所示。

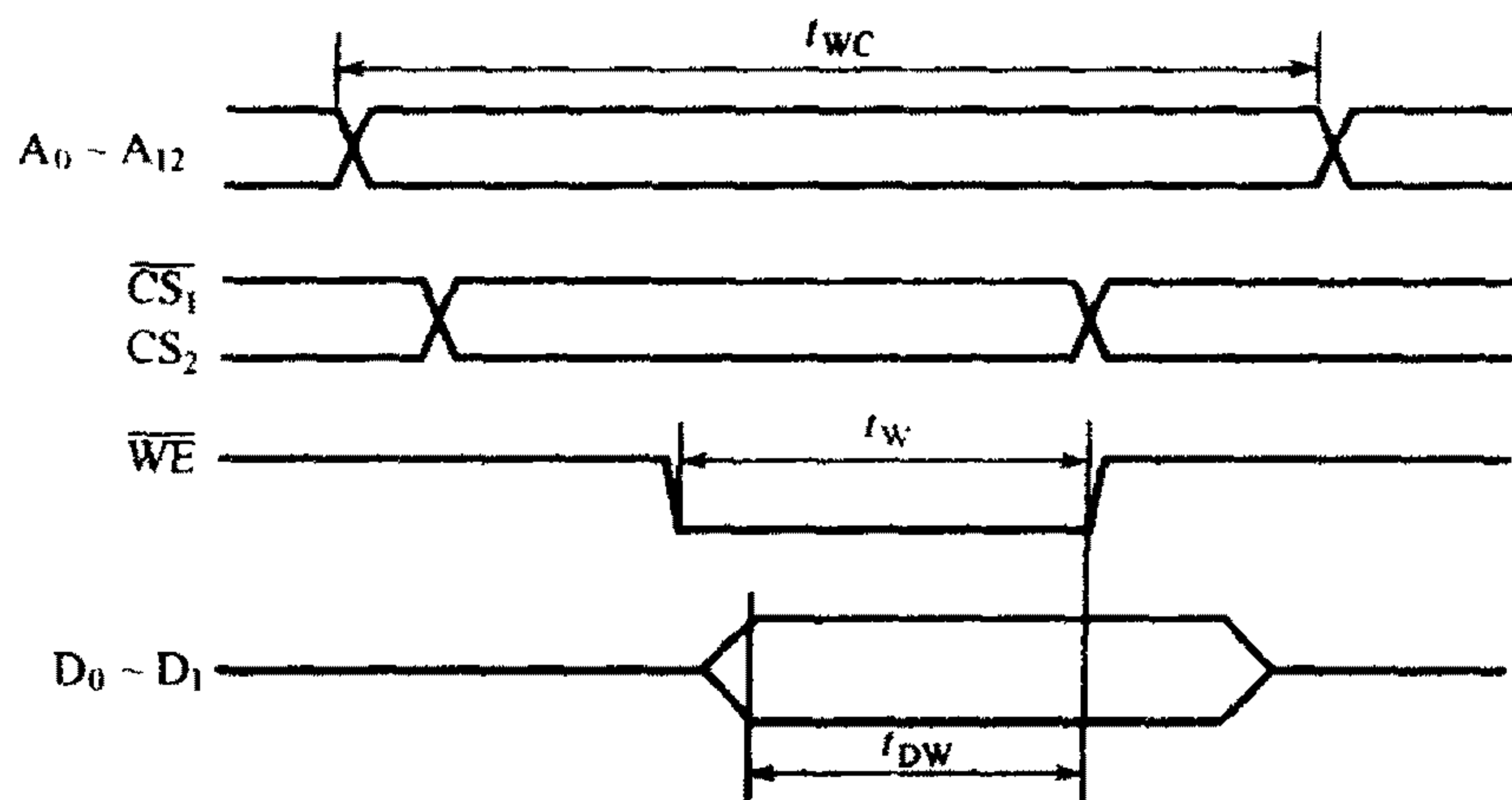


图 6.7 SRAM 6264 写操作时序图

从芯片中读出数据的过程与写操作类似:先把要读出单元的地址送到 6264 的地址线上,然后使 $\overline{CS}_1 = 0$ 和 $CS_2 = 1$ 同时有效;与写操作不同的是,此时要使读允许信号 $\overline{OE} = 0$, $\overline{WE} = 1$,这样,选中单元的内容就可从 6264 的数据线读出。读出过程的时序图略。

主存与 CPU 是两个独立的工作部件,有着完全不同的工作方式和速度,如何协调两者工作速度之间的差异有两种处理方式。一种最简单的解决办法就是降低 CPU 的时钟频率,即延长时钟周期 T_{CLK} ,使 CPU 的工作周期等于内存的存取周期。称为同步方式。但这样做会降低系统的运行速度。另一种方法是利用 CPU 上的 READY 信号,使 CPU 在对慢速存储器操作时插入一个或几个等待周期 T_w ,以等待存储器操作的完成,称之为准同步或半同步方式。这种方式较好地发挥了 CPU 的效率,长期以来为大多数计算机系统所采用。

6264 芯片的功耗很小(工作时为 15 mW,未选中时仅 10 μ W),因此在简单的应用系统中,CPU 可直接和存储器相连,不用增加总线驱动电路。

4. RAM 的译码方式

在了解了 SRAM 芯片的外部引脚功能和它的工作时序之后,更重要的是如何实现它与系统的连接。将一个存储器芯片接到总线上,除部分控制信号及数据信号线的连接外,主要是如何保证该芯片在整个内存中占据的地址范围能够满足系统应用的要求。前面已经说明,芯片的选片信号是由高位地址信号和控制信号的译码产生的,也就是说高位地址信号决定了这个芯片在整个内存中占据的地址范围。用一个形象的例子来解释,若把存储器看成一个居住小区,那么构成存储器的某个存储芯片就是小区内一座居民楼(假定楼号为 01 ~

30), 而存储单元就是楼内的各个居住单元(假定单元号为 101 ~ 825)。若一户居民住在 10 号楼 510 单元, 该地址可以记为 10 - 510, 这里的楼号 10 相当于高位地址, 楼内的单元号 510 就相当于低位地址。要访问小区的 10 - 510 住户时, 首先要找到楼号 10, 这就是片选译码(选择一个存储芯片); 然后再找 510 单元, 这就是片内译码(选择一个存储单元), 片内译码由存储芯片内部完成, 使用者无需考虑。使用者要考虑的只是如何根据地址找到具体的住宅楼(芯片)。

可以说, 存储器芯片应用的精髓就在于存储器的地址译码方法。下面介绍决定芯片存储地址空间的方法和如何实现译码。

1) 地址译码方式

存储器的地址译码方式可以分为两种, 一种称为全地址译码, 另一种称为部分地址译码。

(1) 全地址译码方式

假定系统包括 20 位地址总线, 而构成存储器时要使用全部 20 位的地址总线信号, 即所有的高位地址信号用来作为译码器的输入, 低位地址信号接存储芯片的地址输入线, 从而使得存储器芯片上的每一个单元在整个内存空间中具有惟一的一个地址, 即为全地址译码。

具体到 6264 芯片, 就是用低 13 位地址信号($A_0 \sim A_{12}$)决定每个单元的片内地址, 即片内寻址; 而用高 7 位地址信号($A_{13} \sim A_{19}$)决定芯片在内存中的地址范围, 作为片选地址译码, 如图 6.8 所示。这是一片 SRAM 6264 与 8086/8088 系统的连接图。图中地址总线的高 7 位地

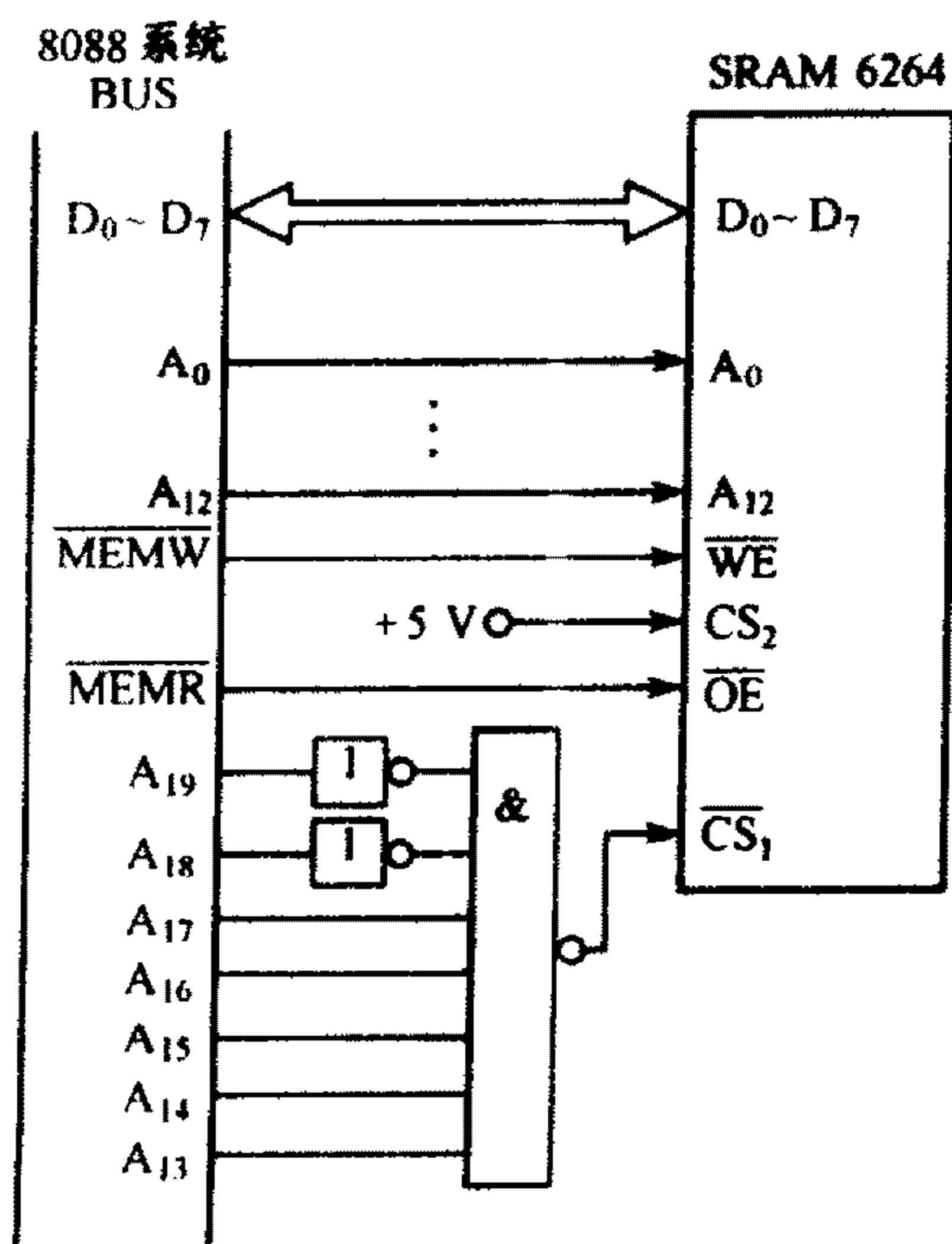


图 6.8 6264 的全地址译码连接

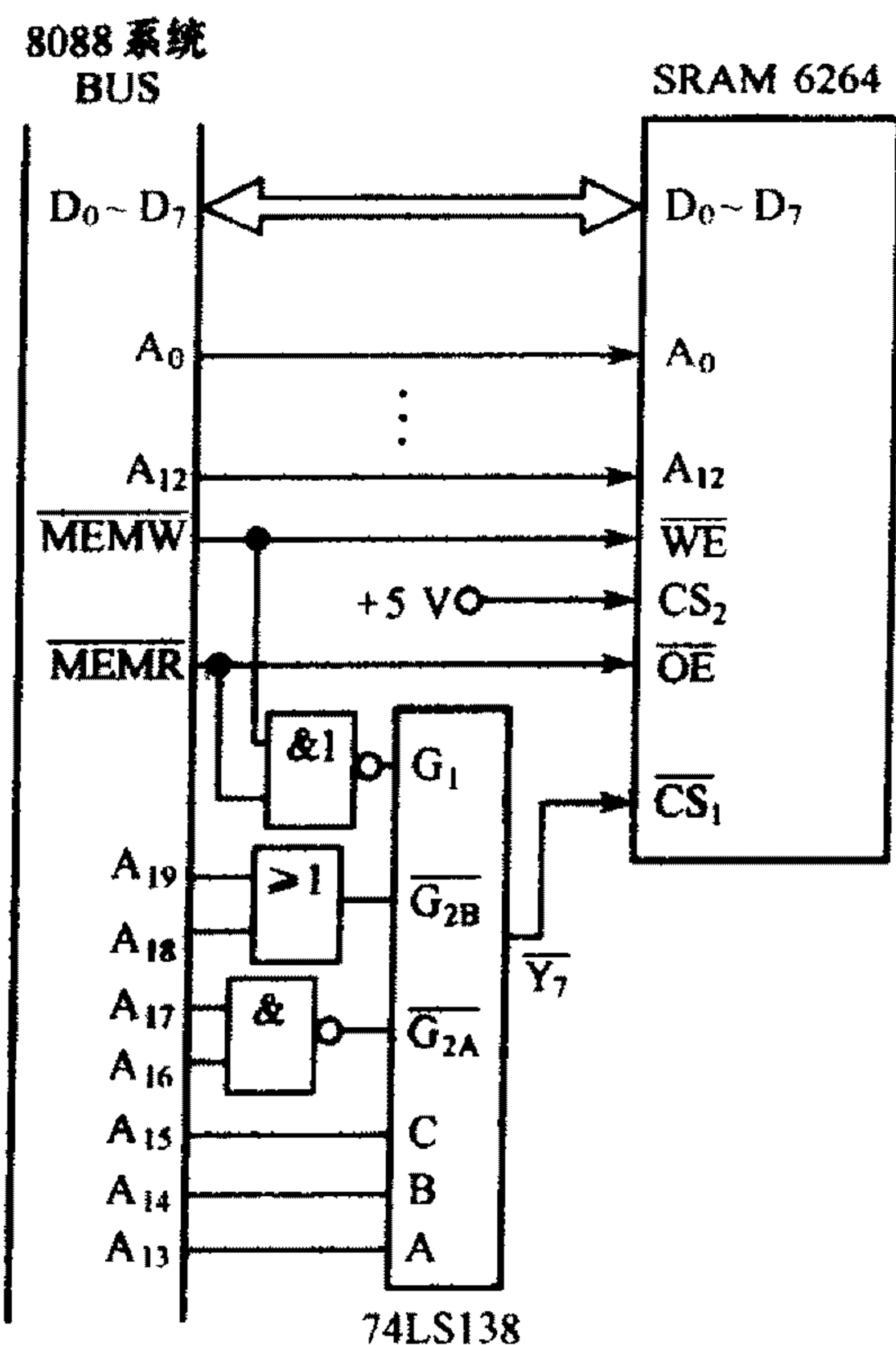


图 6.9 利用 74LS138 译码器实现全地址译码连接

址信号 $A_{13} \sim A_{19}$ 作为地址译码器的输入,地址总线的低 13 位地址信号 $A_0 \sim A_{12}$ 接到芯片的相应的地址引出线端,实现一个全地址译码方式的连接。可以看出,当 $A_{19} \sim A_{13}$ 为 0011111 时,译码器输出为低电平,所以该 6264 芯片的地址范围为 3E000H ~ 3FFFFH(低 13 位可以是全 0 到全 1 之间的任何一个值)。

译码电路的构成不是惟一的,可以利用基本逻辑门电路(如与、或、非门等)构成,也可以利用第 2 章中介绍的 3 线 - 8 线译码器 74LS138 构成。图 6.9 就是用 74LS138 译码器实现同样地址范围的译码电路。

若将图 6.8 中的与非门改为或门,如图 6.10 所示,则 6264 的地址范围就变成 C0000H ~ C1FFFFH。由此可以看出,使用不同的译码电路,可将存储器芯片映射到内存空间中任意一个范围中。

(2) 部分地址译码方式

顾名思义,部分地址译码就是仅把地址总线的一部分地址信号线与存储器连接,通常是用高位地址信号的一部分(而不是全部)作为片选译码信号。图 6.10 就是一个部分地址译码的例子。从图中可以看出,该 6264 芯片被映射到了以下内存空间中:

AE000H ~ AFFFFH

BE000H ~ BFFFFH

EE000H ~ EFFFFH

FE000H ~ FFFFFH

即该 6264 芯片共占据了 4 个 8 KB 的内存空间,而 6264 芯片本身只有 8 KB 的存储容量。为什么会出现这种情况呢?其原因就在于图中的高位地址译码并没有利用地址总线上的全部地址信号,而只利用了其中的一部分。在图 6.10 中, A_{18} 和 A_{16} 并未参加译码,因此, A_{18} 和 A_{16} 无论是什么值都不影响译码器的输出。当 A_{18} 和 A_{16} 分别为 00、01、10、11 这四种组合时,使 6264 这个 8 KB 的存储芯片占据了 4 个 8 KB 的地址空间。这种只用部分地址线参加译码从而产生地址重复区的译码方式就是部分地址译码的含义。按这种地址译码方式,芯片占用的这 4 个 8 KB 的区域决不可再分配给其他芯片。否则,会造成总线竞争而使处理器无法正常工作。另外,在对这个 6264 芯片进行存取时,可以使用以上 4 个地址范围的任意一个。

部分地址译码使地址出现重叠区,而重叠的部分必须空着不准使用,这就破坏了地址空间的连续性,实际上就是减小了总的可用存储地址空间。部分地址译码方式的优点是其译码器的构成比较简单,成本较低。图 6.11 中就少用两条译码输入线,但这是以牺牲可用内存空间为代价换来的。

显然,参加译码的高位地址越少,译码器就越简单,而同时所构成的存储器占用的内存

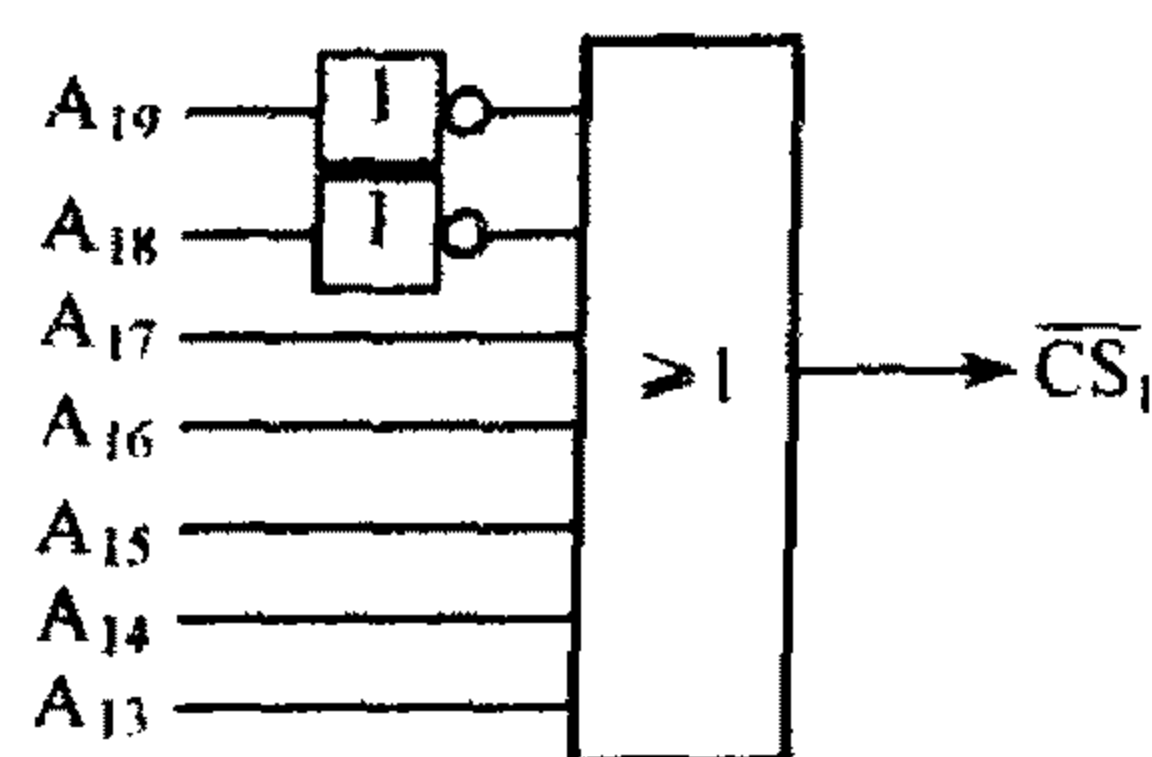


图 6.10 另一种译码电路

地址空间就越多。若只用一条高位地址线做片选信号,如在图 6.10 中,若只将 A_{19} 接在 \overline{CS}_1 上,则这片 6264 芯片将占据 00000H ~ 7FFFFH 共 512 KB 的地址空间。这种只用一条高位地址线进行选片的连接方法称为线性选择,这种地址译码方法一般仅在系统中只使用 1~2 个存储芯片时可考虑使用。

在实践中,采用全地址译码还是部分地址译码,应根据具体情况来定。如果地址资源很富裕,为使电路简单可考虑用部分地址译码方式。如果要充分利用地址空间,则应采用全地址译码方式。

2) 静态 RAM 的应用举例

以上讲述了当利用 RAM 芯片构成内存时常采用的两种地址译码方式,其中最常使用的是全地址译码。在上面已经提到,实现全地址译码可以使用各种基本逻辑门电路,也可以用现成的译码器芯片,如 74LS138 译码器等。(译码器的种类很多,如其他 74 系列芯片、PAL、GAL 等,限于篇幅这里就不一一介绍了)。下面通过一个例子来介绍如何用 SRAM 芯片构成所需的存储器。

例 6-1 用存储器芯片 SRAM 6116 构成一个 4 KB 的存储器。要求其地址范围在 78000H ~ 78FFFH 之间。

图 6.12 所示是 6116 芯片的外部引脚图。由图可知,6116 芯片有 11 根地址线($A_0 \sim A_{10}$),8 根数据线($D_0 \sim D_7$),读/写控制信号 R/\overline{W} (当 $R/\overline{W} = 0$ 时写入, $R/\overline{W} = 1$ 时读出),输出允许信号 \overline{OE} 及片选信号 \overline{CS} 。由芯片的地址线和数据线的根数可以很容易地看出,6116 为 $2\text{ K} \times 8\text{ bit}$ 的存储芯片。因此,要构成一个 4 KB 的存储器,则需要两片 6116 芯片。

这里,选用 74LS138 作为地址译码器,采用全地址译码方式,使两片 6116 具有惟一的地址范围。图 6.13 所示为存储器与 8088 微型计算机系统总线的连接图。图中,用 74LS138 和一些门电路构成地址译码器,对地址线高 9 位($A_{11} \sim A_{19}$)进行译码。将 \overline{MEMR} 、 \overline{MEMW} 信号组合后接到 74LS138 译码器的使能端,保证了仅在对存储器进行读/写操作时,74LS138 译码器才能工作。

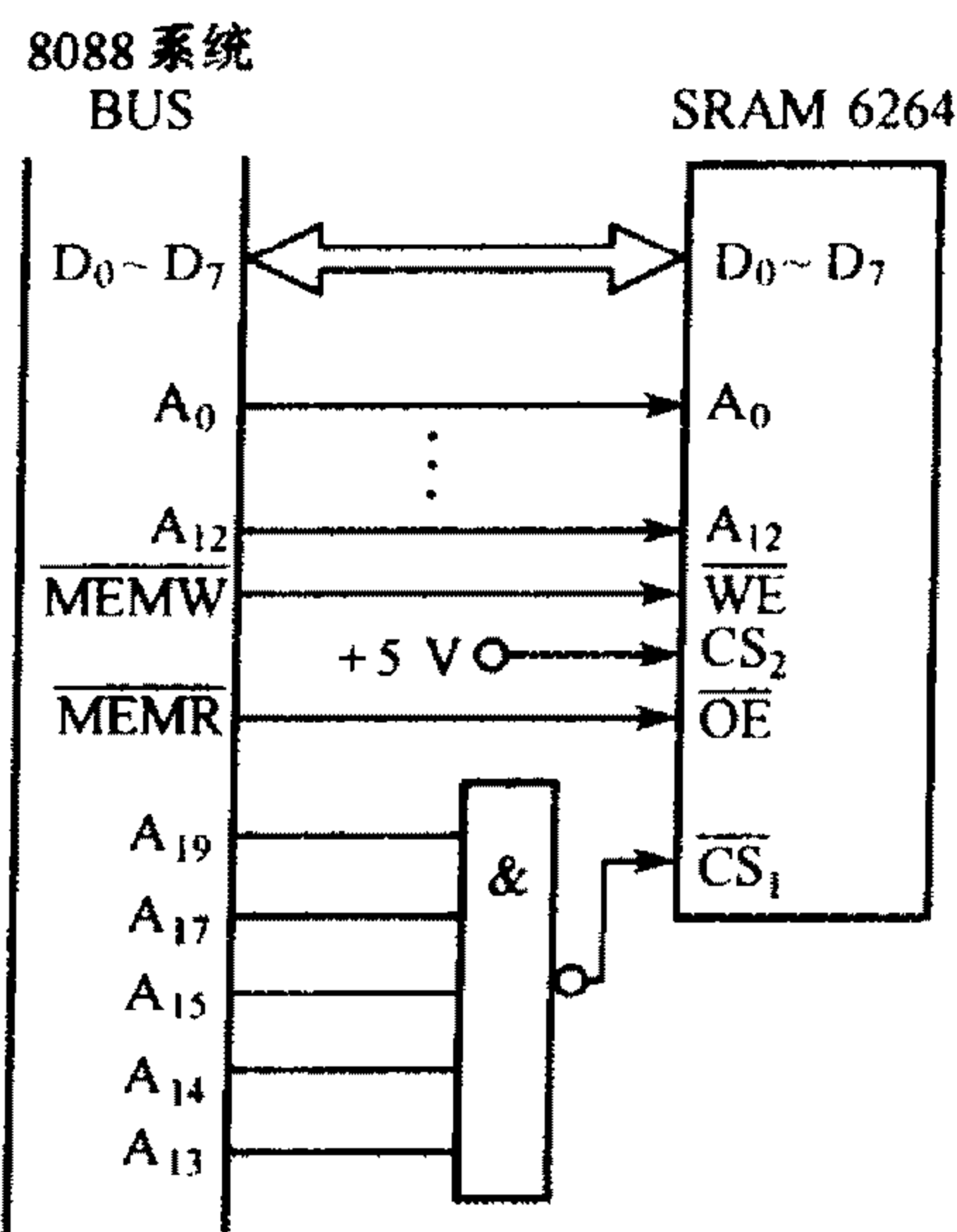


图 6.11 SRAM 6264 的部分地址译码连接图

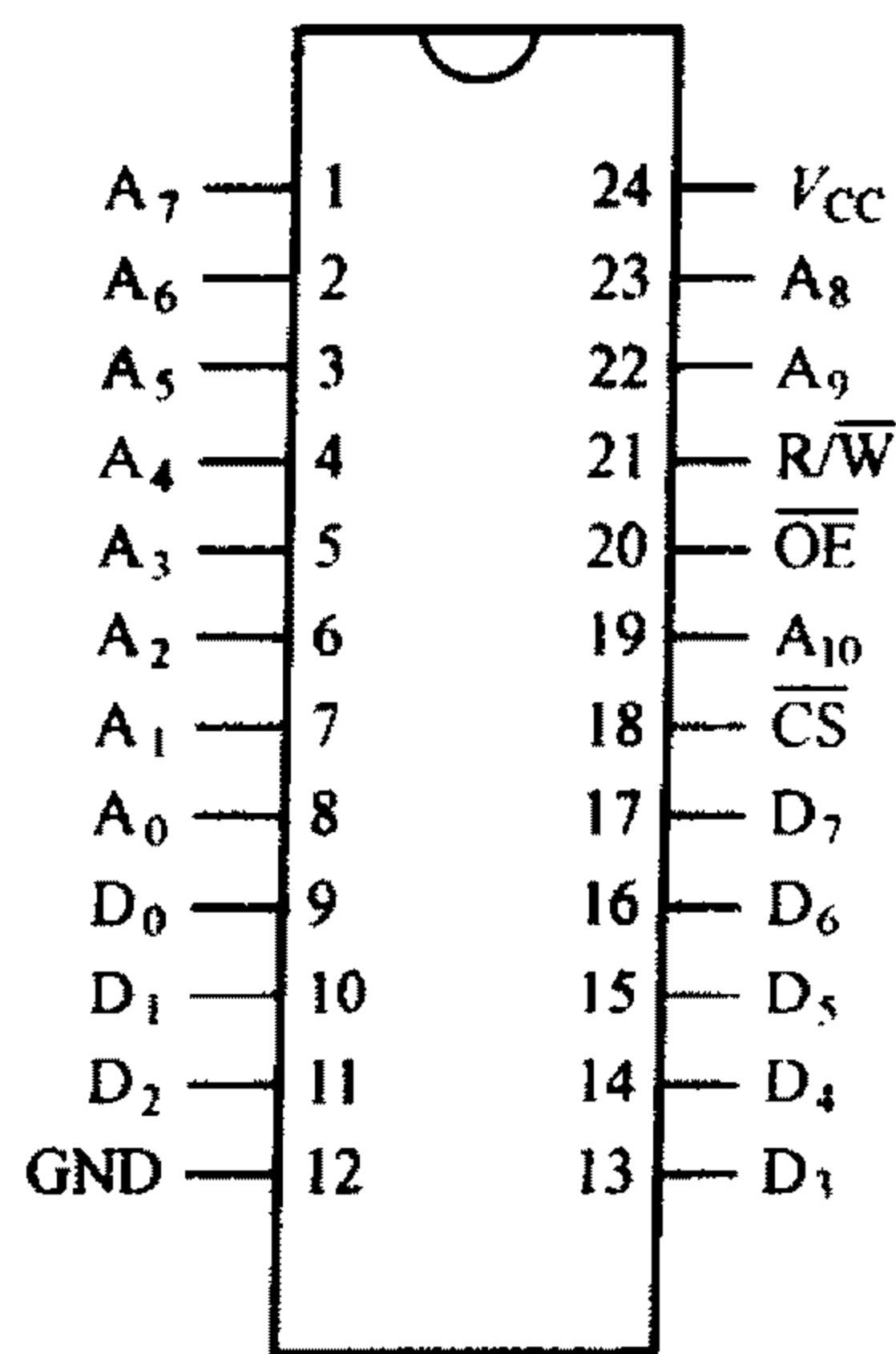


图 6.12 SRAM 6116 的引脚图

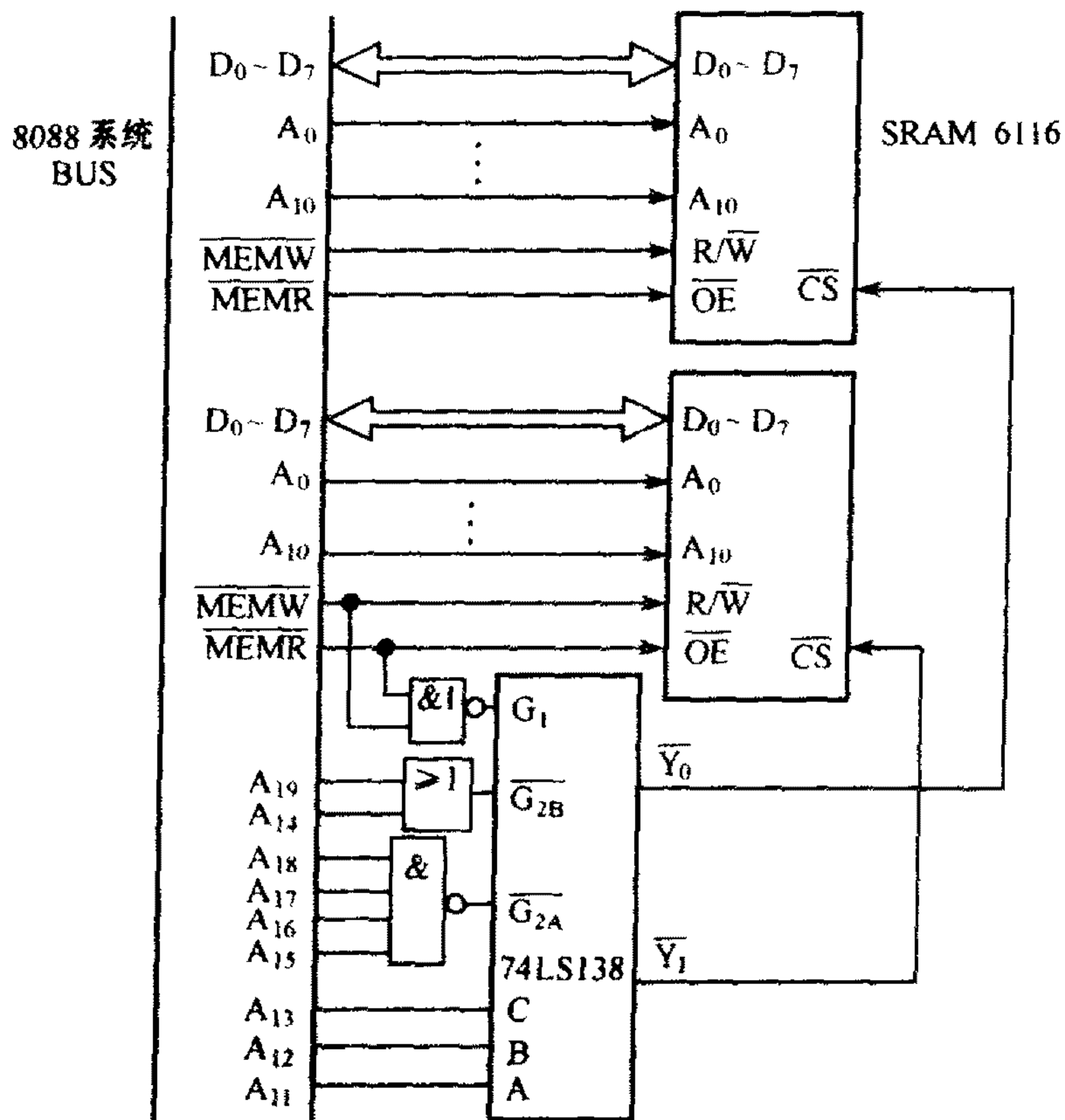


图 6.13 6116 的应用连接图

以上的例子可以看出,在用 SRAM 芯片构造存储器时,可以采用多种连接方式。只要了解了 SRAM 芯片的外部引出线含义,再根据 CPU 总线所能提供的信号,选择适当的器件构成译码器,便可以很容易地构成所需的存储器空间。

6.2.3 动态随机存储器(DRAM)

1. DRAM 的工作原理

动态 MOS 存储元靠电容存储信息,电容充有电荷表示存储信息“1”,电容放电表示存储“0”。基本存储元由一个 MOS 管和一个与源极相连的电容 C_s 组成。如图 6.14 所示。其漏极连接位线(数据线)D,栅极接字线 W。

写入时,字线加高电平, T_1 导通。若数据线为高电平,则对 C_s 充电,写入信息“1”;若数据线为低电平, C_s 通过 T_1 放电,写入信息“0”。

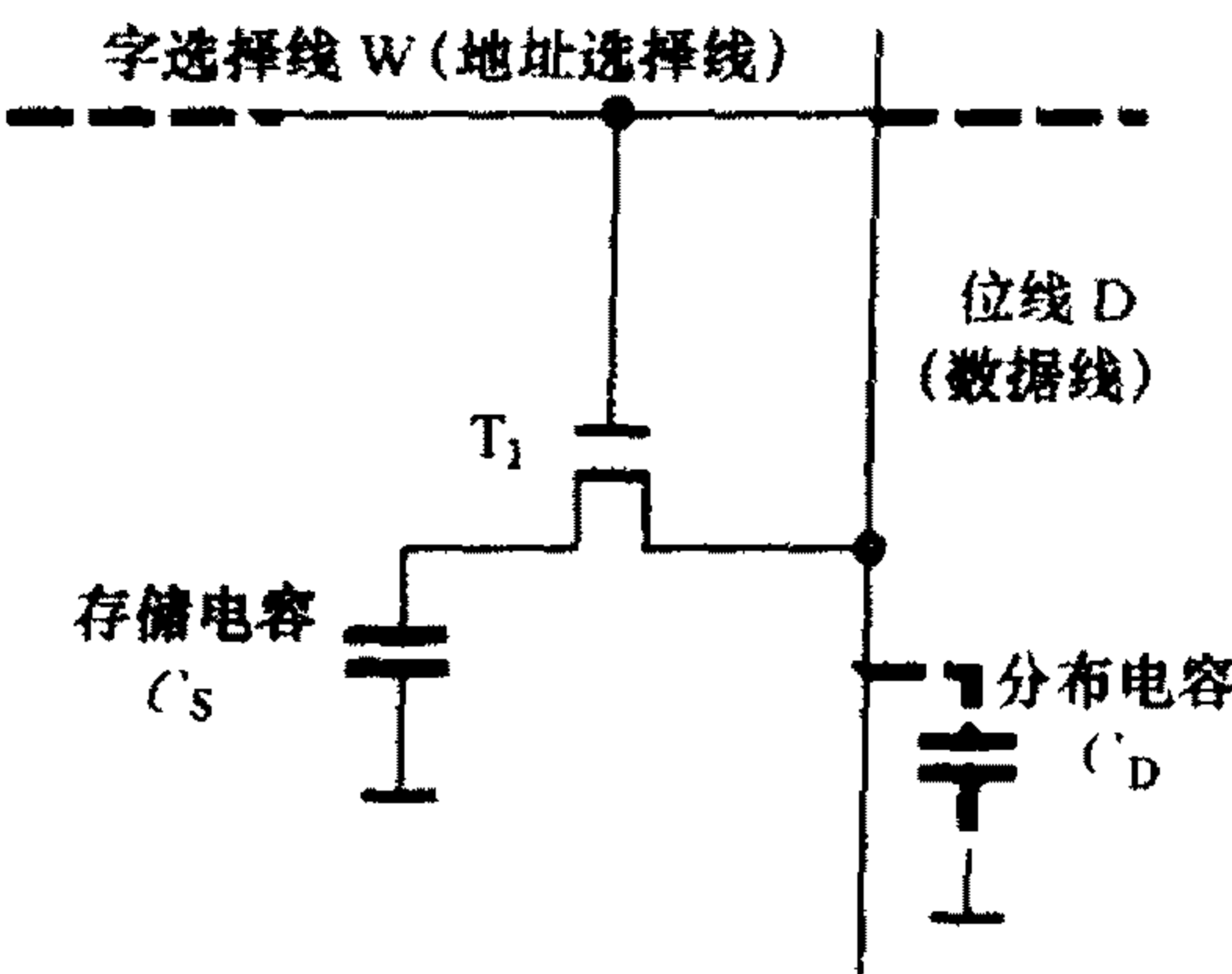


图 6.14 单管动态存储元的电路

暂存时,字线加低电平, T_1 断开,电容 C_s 基本无放电回路,可暂时保存电荷或维持无电荷的“0”状态。

读出时,数据线预充电,当字选择线为高电平时, T_1 导通,存储在 C_s 电容上的电荷通过 T_1 输出到位线上;若原来 C_s 上无电荷,则数据线上无电位变化,根据位线有无电流即可得知存储的信息是“1”还是“0”。

DRAM集成度高、价格低,在微型计算机中有着极其广泛的使用。构成微型计算机内存的内存条几乎毫无例外地都是由DRAM组成的。它的缺点是:

- 1) 读出是破坏性的,故读出后要立即进行“重写”;
- 2) 存储时由于有漏电流, C_s 上的电荷会泄漏,需要周期性地给存储“1”的电容补充电荷,即“刷新”,才能维持信息。

2. DRAM 的外部特性及工作过程

以一种DRAM芯片2164A为例来说明。

1) 2164A 的引脚功能

2164A是一块 $64K \times 1\text{bit}$ 的DRAM芯片,与其类似的芯片有很多种,如3764、4164等。图6.15所示为2164A的引脚图。

$A_0 \sim A_7$: 地址输入线。DRAM芯片在构造上的特点是芯片上的地址引线是复用的。虽然2164的容量为64 K个单元,但它并不像对应的SRAM芯片那样有16根地址线,而是只有这个数量的一半,即8根地址线。在存取DRAM芯片的某单元时,其操作过程是将存取的地址分两次输入到芯片中,每一次都由同一组地址线输入。两次送到芯片上去的地址分别称为行地址和列地址。它们被锁存到芯片内部的行地址锁存器和列地址锁存器中。

可以想像到在芯片内部,各存储单元是按照矩阵结构排列的。行地址信号通过片内译码选择一行,列地址信号通过片内译码选择一列,这样就决定了选中的单元。可以简单地认为该芯片有256行和256列,共同决定64 K个单元。对于其他DRAM芯片也可以按同样方式考虑。如21256,它是 $256K \times 1\text{bit}$ 的DRAM芯片,有256行,每行为1024列。

因此,动态存储器芯片上的地址引线是复用的,CPU对它寻址时的地址信号分成行地址和列地址,分别由芯片上的地址线送入芯片内部进行锁存、译码,从而选中要寻址的单元。

D_{IN} 和 D_{OUT} : 芯片的数据输入和输出线。其中, D_{IN} 为数据输入线,当CPU写芯片的某一单元时,要写入的数据由 D_{IN} 送到芯片内部; D_{OUT} 是数据输出线,当CPU读芯片的某一单元时,数据由此线输出。

\overline{RAS} : 行地址锁存信号。该信号将行地址锁存在芯片内部的行地址锁存器中。

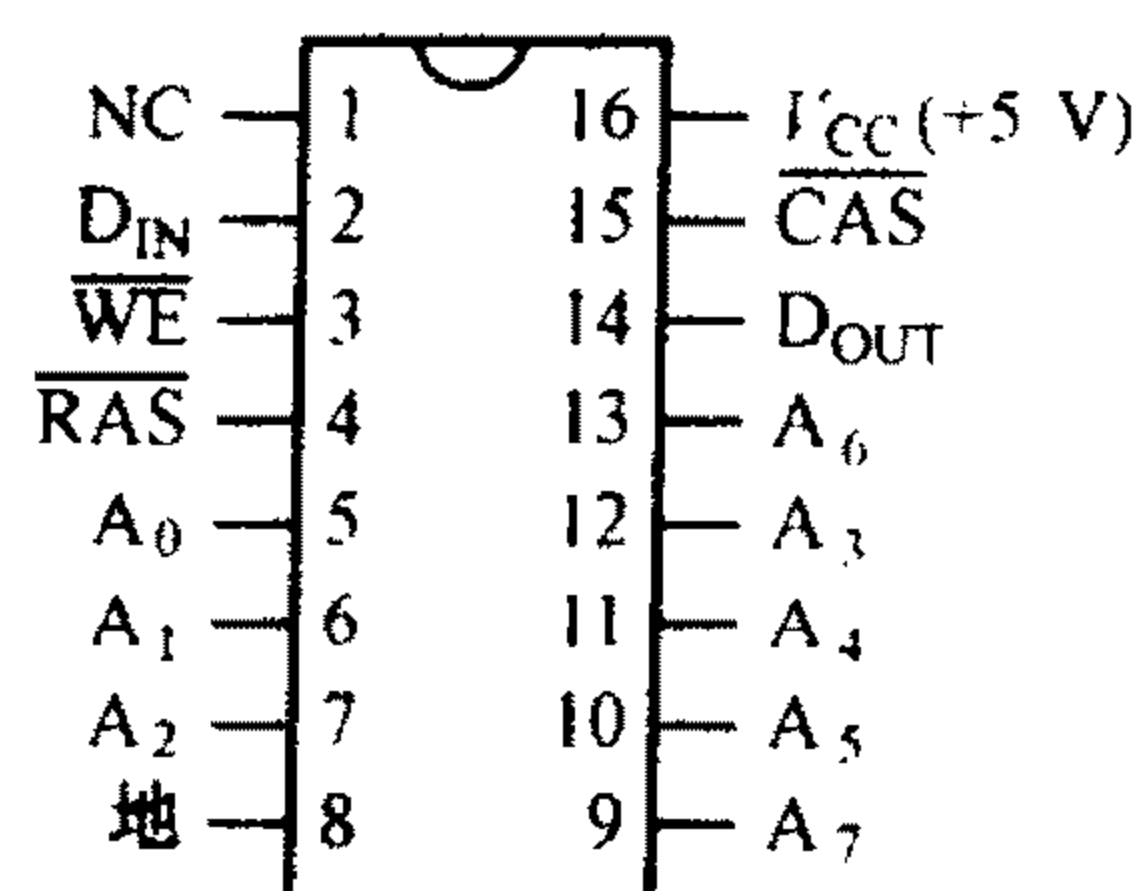


图 6.15 2164A 外部引脚图

$\overline{\text{CAS}}$: 列地址锁存信号。该信号将列地址锁存在芯片内部的列地址锁存器中。

$\overline{\text{WE}}$: 写允许信号。当 $\overline{\text{WE}} = 0$ 时,允许将数据写入;反之,当 $\overline{\text{WE}} = 1$ 时,可以从芯片读出数据。

2) 工作过程

(1) 数据读出

读出过程的时序如图 6.16 所示。首先将行地址加在 $A_0 \sim A_7$ 上,然后使 $\overline{\text{RAS}}$ 行地址锁存信号有效,该信号的下降沿将行地址锁存在芯片内部。接着将列地址加到芯片的 $A_0 \sim A_7$ 上,再使 $\overline{\text{CAS}}$ 列地址锁存信号有效,该信号的下降沿将列地址锁存在芯片内部。然后保持 $\overline{\text{WE}} = 1$,则在 $\overline{\text{CAS}}$ 有效期间(低电平),数据由 D_{OUT} 端输出并保持。

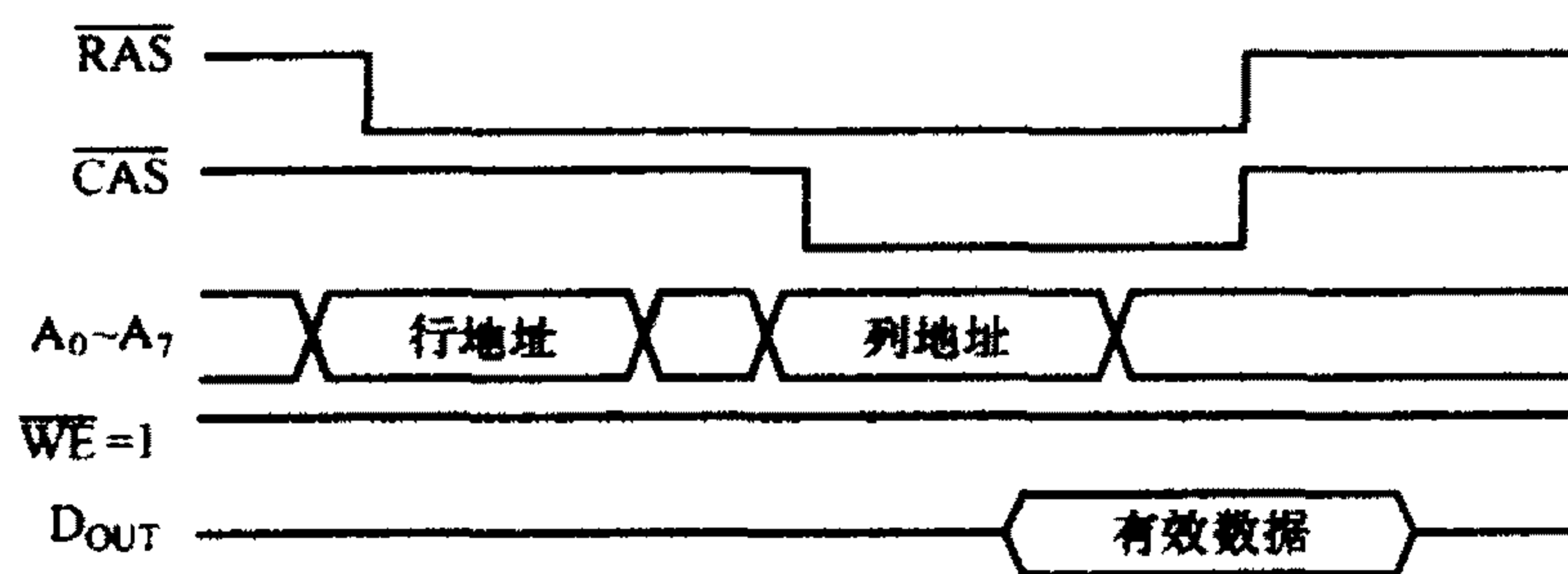


图 6.16 DRAM 2164 的数据读出时序图

(2) 数据写入

数据写入过程的时序如图 6.17 所示。数据写入与数据读出的过程基本类似,区别是送完列地址后,要将 $\overline{\text{WE}}$ 端置为低电平,然后把要写入的数据从 D_{IN} 端输入。

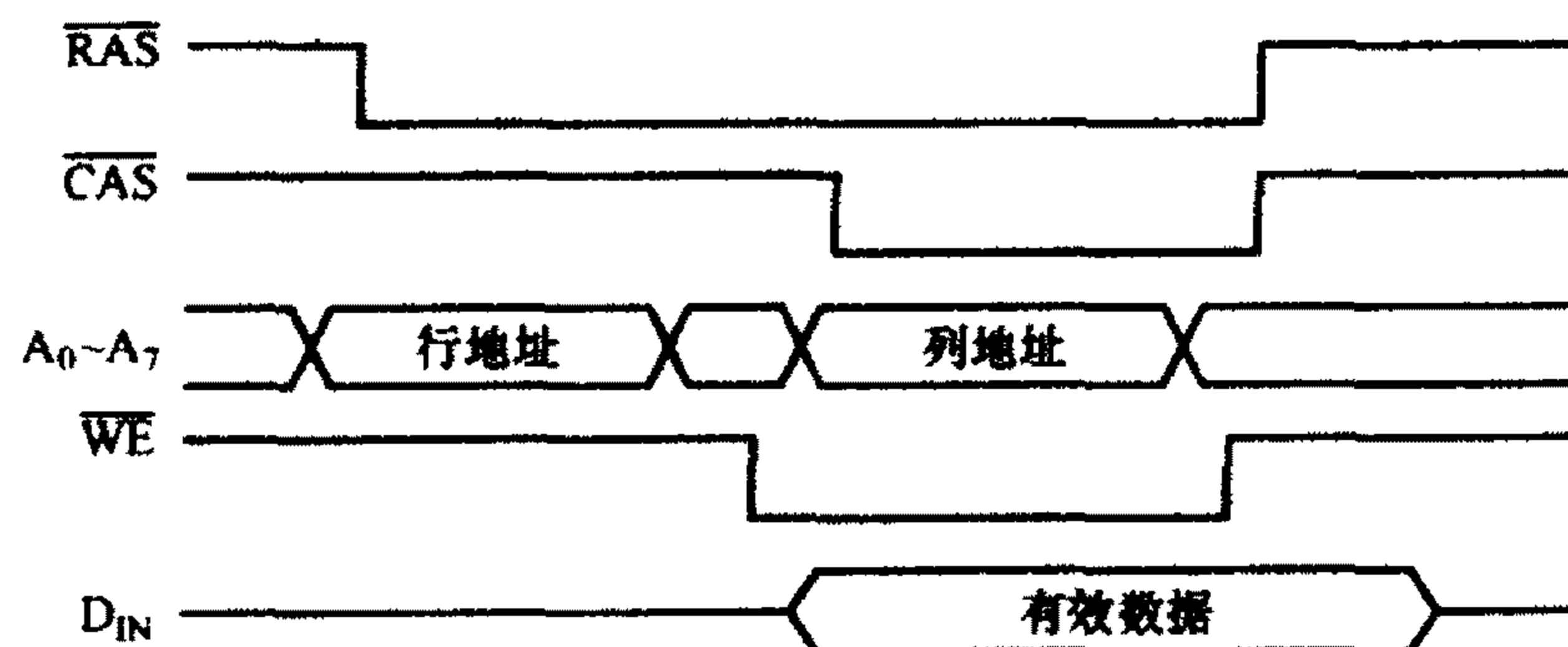


图 6.17 DRAM 2164 的数据写入时序图

(3) 刷新

DRAM 芯片的刷新时序如图 6.18 所示。图中 $\overline{\text{CAS}}$ 保持无效,利用 $\overline{\text{RAS}}$ 锁存刷新的行地址,进行逐行刷新。DRAM 要求每隔 2 ~ 8 ms 刷新一次,这个时间称为刷新周期。在刷新周

期中, DRAM 是不能进行正常的读/写操作的, 这一点由刷新控制电路来予以保证。

3. DRAM 的刷新方式

DRAM 采用“读出”方式进行刷新。因为在读出过程中恢复了存储元的 MOS 栅极电容电荷, 所以, 读出过程即刷新过程。一般地, 在刷新过程中只改变行

选择线地址, 每次刷新一行, 依次对存储器的每一行进行读出, 就可完成对整个 RAM 的刷新。全部刷新一遍所允许的最大时间间隔, 称为最大刷新周期。刷新方式通常有以下三种。

1) 集中式刷新(Burst Refresh)

集中式刷新指在一个刷新周期内, 利用一段固定的时间, 依次对存储器的所有行逐一刷新, 在此期间停止对存储器的读/写。例如, 一个存储器系统的最大刷新周期为 2 ms, 在此期间集中安排若干个刷新周期, 其周期数等于最大容量芯片的行数, 使全部芯片刷新一遍; 其余时间可正常读/写, 如图 6.19 所示。此种工作方式的优点是主存利用率高, 控制简单。缺点是刷新状态不能读/写主存储器, 因而形成一段“死区”。

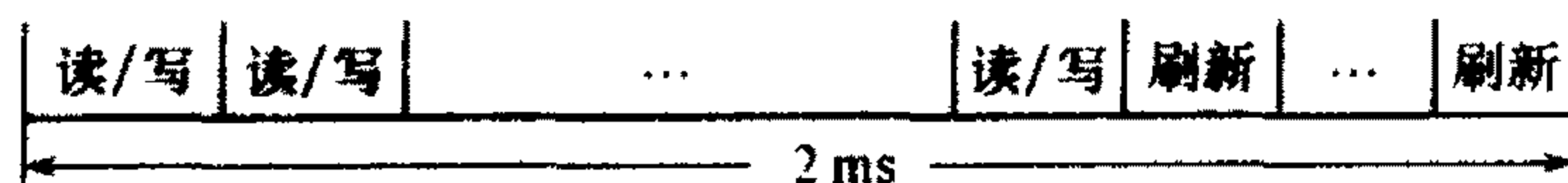


图 6.19 集中式刷新安排方式示意图

2) 分布式刷新(Distributed Refresh)

分布式刷新把每一行的刷新周期分散到各个读/写周期中去, 这样, 一个存储器的工作周期分为两部分: 前半期用于正常读/写或保持, 后半期用于刷新某一行。分布式刷新的好处是控制简单, 主存工作没有长的死区。缺点是主存利用率低, 工作速度大约降低一倍。如图 6.20 所示。

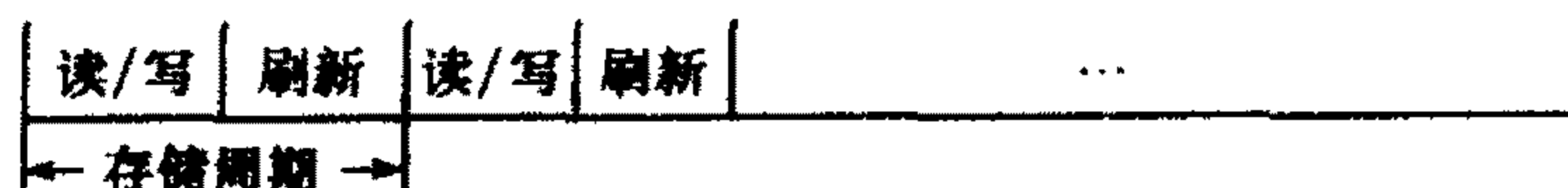


图 6.20 分布式刷新安排方式示意图

3) 异步刷新(Asynchronous Refresh)

异步刷新按芯片的行数决定所需要的刷新周期数, 并分散安排在最大刷新周期之中。如图 6.21 所示, 每隔若干时间片提出一次刷新请求, 响应后即开始一个刷新周期。如果发

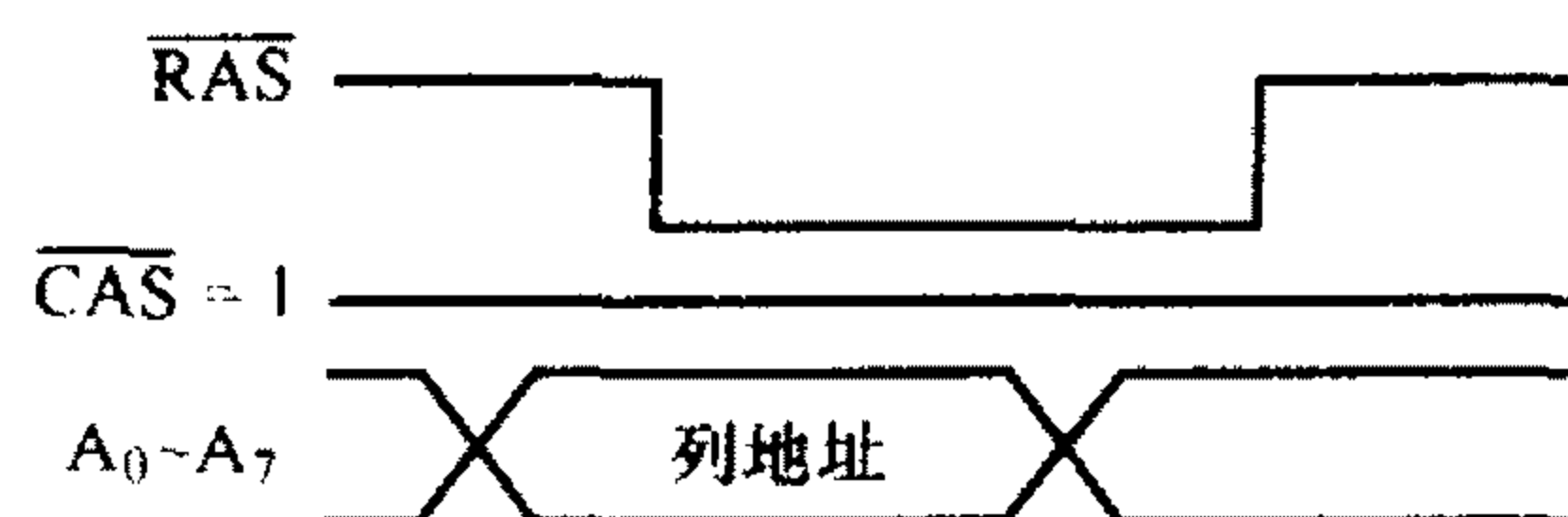


图 6.18 DRAM 芯片的刷新时序图

出刷新请求时 CPU 读/写尚未结束,则等待主存空闲再安排刷新周期,故称为异步刷新方式。目前大多数计算机系统都利用 DMA 方式(直接内存访问)发出刷新请求, CPU 响应后让出系统总线,由 DMA 控制器接管总线控制权,送出刷新地址进行一次动态刷新。

异步刷新方式没有死区,对主存利用率和速度两者影响最小,可以说综合了以上两种方式的优点。虽然控制较复杂,但能够利用系统已具备的 DMA 功能来实现。因此一般计算机系统都采用异步刷新方式。

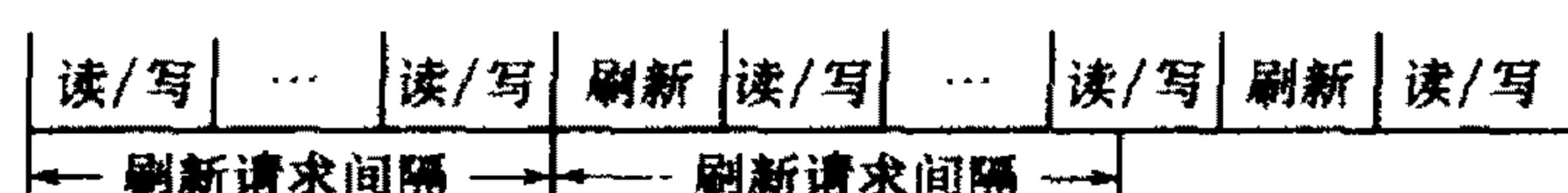


图 6.21 异步刷新安排方式示意图

4. DRAM 在系统中的连接

由于 DRAM 在使用中既要做到能够正确读/写,又要能在规定时间里进行刷新。因此计算机中对 DRAM 的连接和控制电路要比 SRAM 复杂得多。图 6.22 所示的是 PC/XT 微型计算机的 DRAM 读/写简化电路图。读者可通过它来简单了解 DRAM 的使用。

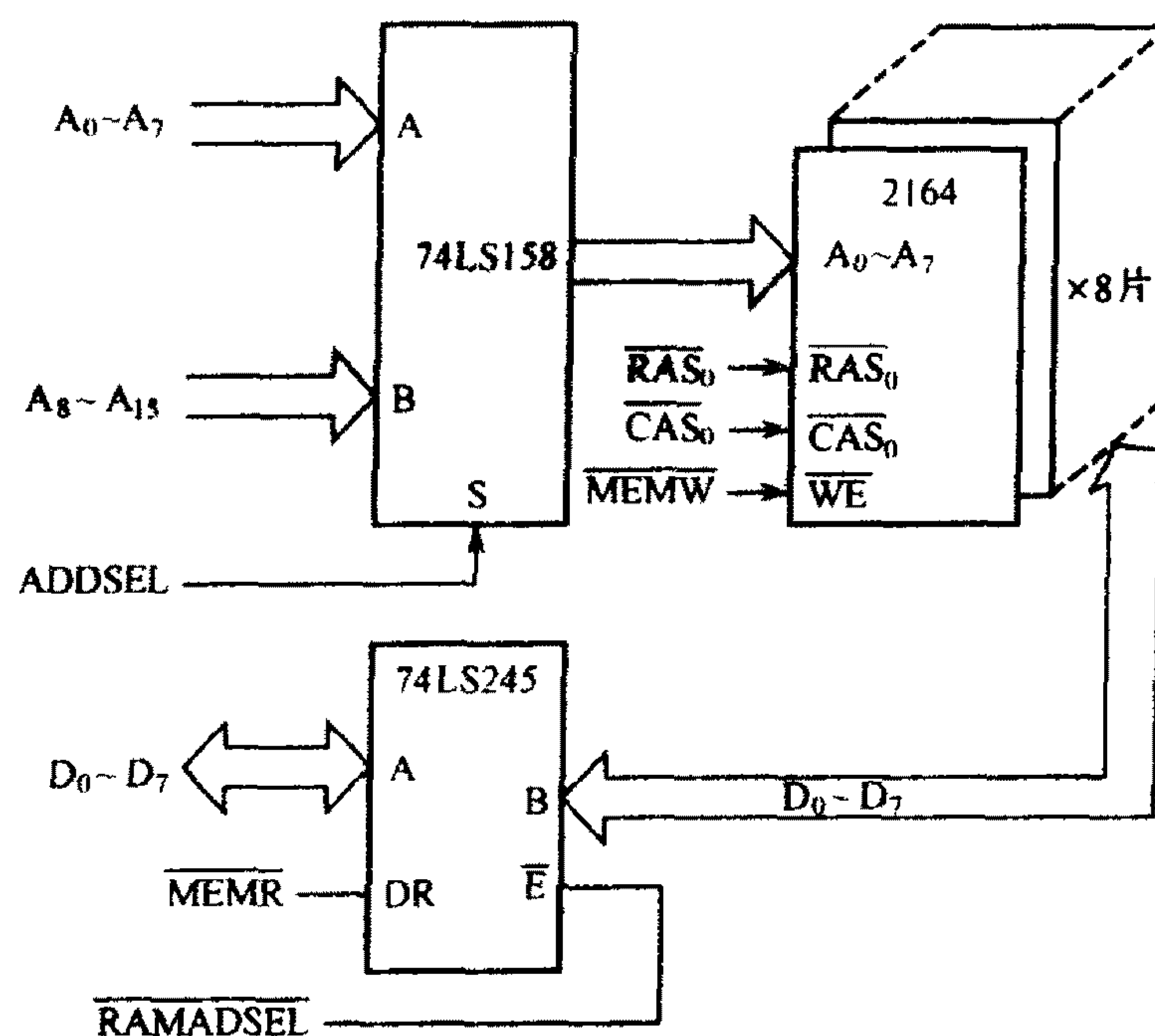


图 6.22 DRAM 读/写简化电路示意图

图中用虚线画的长方体表示由 8 片(加奇偶校验位则为 9 片)2164 DRAM 组成的 64 KB 的存储器。74LS158 是二选一的数据选择器,74LS245 为驱动器。当 CPU 读/写存储器的某

个单元时,首先由行/列锁存信号电路送出行地址锁存信号 $\overline{\text{RAS}}$,同时 $\text{ADDSEL} = 0$,使74LS158的A端口导通,CPU将8位行地址信号通过地址总线的低8位($A_0 \sim A_7$),再通过74LS158的A口加到存储器芯片上,并在 $\overline{\text{RAS}}$ 作用下锁存于存储芯片内部的行地址锁存器。60 ns后, $\text{ADDSEL} = 1$,使74LS158的B端口导通,CPU将8位列地址信号通过地址总线的 $A_8 \sim A_{15}$,再通过74LS158的B口加到存储器芯片上,延迟40 ns后由 $\overline{\text{CAS}}$ 将其锁存于存储芯片内部的列地址锁存器。最后,在存储器读/写信号 $\overline{\text{MEMR}}/\overline{\text{MEMW}}$ 控制下,实现数据的读/写。

PC/XT微型计算机中DRAM的刷新过程是利用DMA来实现的。首先由可编程定时器8253每隔 $15.12 \mu\text{s}$ 产生一次DMA请求,之后由DMA控制器8237在其 $\overline{\text{DAK0}}$ 端产生一个低电平,使列地址锁存信号 $\overline{\text{CAS}}$ 为高电平,而行地址信号 $\overline{\text{RAS}}$ 为低电平。最后,通过DMA控制器送出刷新的行地址,实现一次刷新。

这里需要说明的是,在目前,以上介绍的几种SRAM和DRAM芯片已不在大多数通用计算机中使用,但在现代生活中,除了通用计算机以外,以单片机为核心的大小各类智能化系统已遍布社会各个领域。作为某些单片机的外围存储器,6116、6264等DIP芯片仍有市场。书中介绍这些传统存储器及其应用连接方式,旨在于让读者通过简单清晰的结构图形了解存储器的工作原理和外部特性,并掌握用简单的存储器芯片构成存储空间的方法。

6.3 只读存储器(ROM)

在计算机中需要把信息存入存储器或从其中读出,但在有些场合要求存储的是固定信息,如系统的基本输入/输出指令、微程序、特殊编码(如字符发生器、汉字库)等。对于这种在使用时只读出已存入的信息,而不能写入新的信息的存储媒体称为只读存储器(ROM)。由于ROM工作时只是读出信息,可以通过设置或不设置半导体管、熔丝等元件来表示存入的二进制信息,因此它的结构比RAM简单。广泛使用的半导体存储器有以下几种类型。

6.3.1 掩模型只读存储器(MROM)

MROM是由生产厂生产的存有固定信息的ROM(Mask Read Only Memory),在制造之前先由用户提供所需固化的信息,以0、1代码表示。芯片生产厂据此设计相应的光刻掩模。

6.3.2 一次编程型只读存储器(PROM)

PROM又称为可编程型存储器,因为这种芯片在封装出厂时内容全为“0”,用户可用专门的写入器写入信息,即将某些存储位写入“1”,一经写入,则不可改变。常用的PROM根据

写入方式可分为结破坏型和熔丝型。

1) 结破坏型

在行、列的交点处焊接一对彼此反向的二极管,它们因彼此反向而不能导通,作为“0”信息。若要该位写入1,则在相应的交点处加高电压,将反偏的一只二极管击穿,只留下正向导通的二极管,作为信息“1”,显然,这就永久性地记录了所写入的信息。

2) 熔丝型

如图 6.23 所示,图中是一个 4 个发射极的存储阵列(4 行 4 列),在行、列的交点连接一段熔丝,有熔丝的位即为“0”。当需要写入“1”时,则通过较大电流使熔丝熔断。图示阵列已存入信息,从 0[#] ~ 3[#] 单元依次为: 0110, 1011, 1010, 0101。读出过程是: 地址输入后经行译码选择某一行线(字线),即为某一编址单元。列线输出读出的信息,即各位线输出。

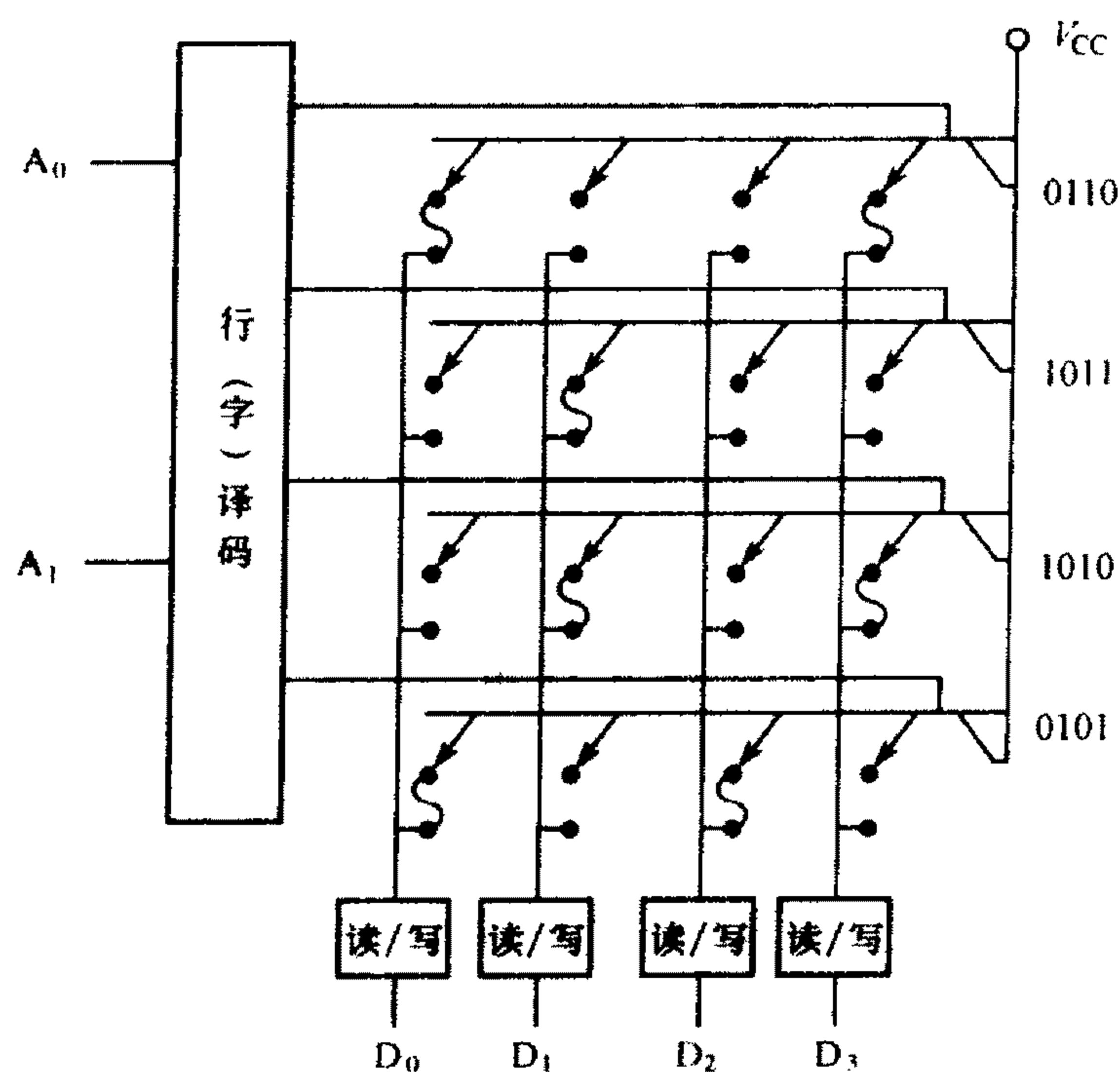


图 6.23 熔丝型 PROM 原理图

6.3.3 可重写只读存储器 (EPROM)

1. EPROM 的工作原理

EPROM(Erasable PROM)是一种可擦除可编程的只读存储器。写入时,用专门的写入器在 +25 V 高压下写入信息。擦除时,用紫外线照射芯片上的窗口,即可清除存储的内容,然后可对其重新编程(写入新的内容)。正常使用时,存储在 EPROM 中的内容能够长期保存达

几十年之久,一般可重写数十次。图 6.24 为 P 沟道 MOS 管 EPROM 的结构示意图。它在 N 型基片上制造两个高浓度的 P 型区,分别引出源极 S 和漏极 D,栅极浮置在氧化硅绝缘层中,没有引出线,称为浮栅。

写入信息前,浮栅上无电荷积存,管内没有导电沟道,因此在 +5 V 电源下 S 与 D 不导通,一般作为“1”状态。写入信息时,将所选单元的源极和漏极之间加 +25 V 高压, S 为正, D 为负。浮栅和硅基片之间的绝缘层很薄,在漏极附近的强电场作用下, D 与浮栅间发生瞬间雪崩击穿,大量电子注入浮栅。撤除高压后,绝缘层又恢复绝缘状态,浮栅上的电子能量不足,不能使电子穿过绝缘层,故浮栅上的电子积存起来。由于浮栅上带负电荷,在硅基片上对应一侧形成带正电荷的 P 沟道,如图 6.24 所示状态。在源极与漏极之间加工作电压 +5 V,使 MOS 管导通,通常定义为“0”状态。以上说明,对 EPROM 写入的过程,是将某些位单元由 1 改写为 0。

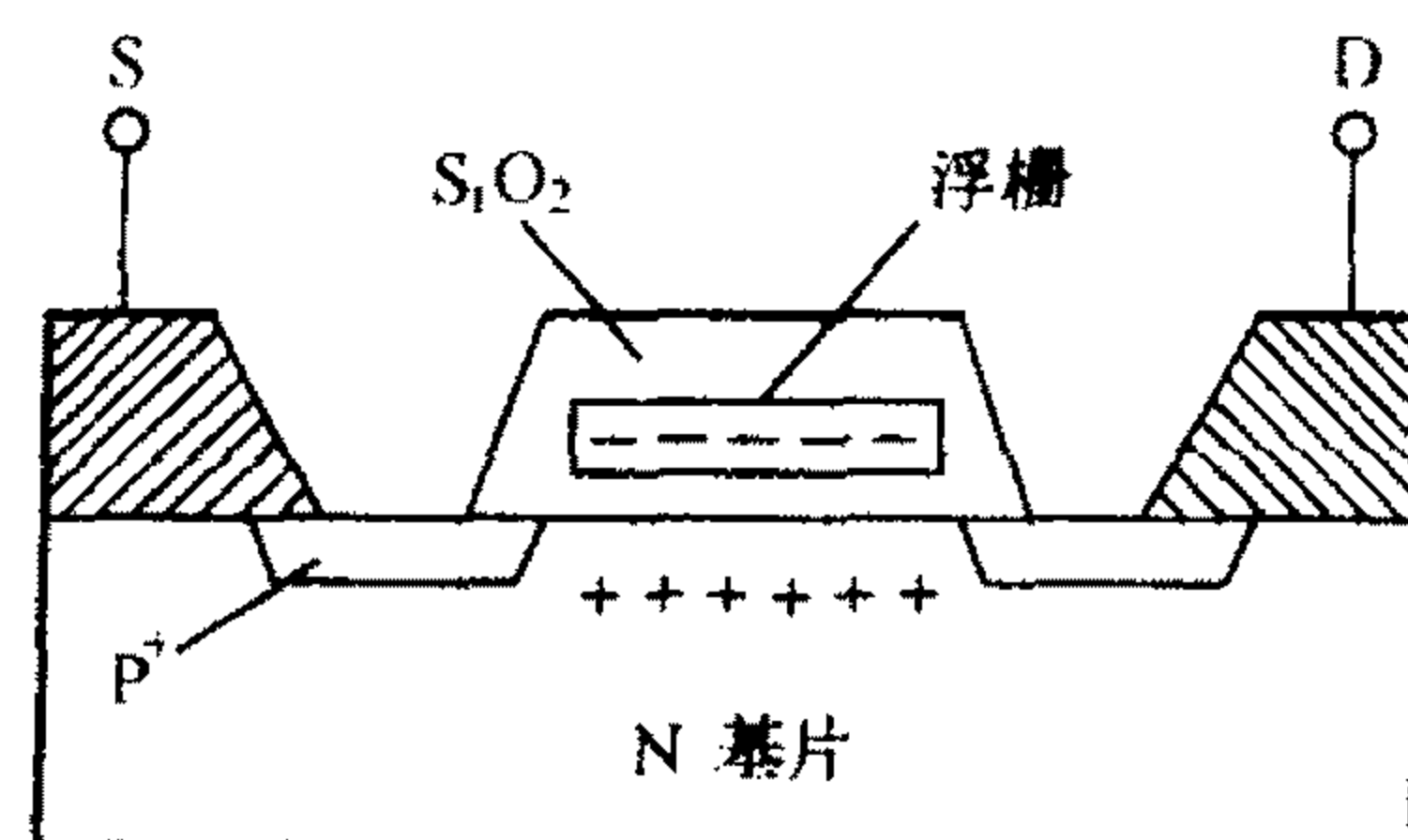


图 6.24 P 沟道 EPROM 结

擦除时,浮栅上的电子因紫外线照射而获得较高的能量,从而会穿过绝缘层泄掉。失去电子的位单元被擦除为 1。

2. 典型 EPROM 2764 的特点和工作方式

1) 2764 的引脚功能

2764 的外部引脚如图 6.25 所示。这是容量为 8 K × 8 bit 的 EPROM 芯片,它的引脚与前面介绍的 SRAM 芯片 6264 是兼容的。这给使用者带来很大方便。因为在软件调试过程中,程序经常需要修改,此时可将程序先放在 6264 中,读/写修改都很方便。调试成功后,将程序固化在 2764 中,由于它与 6264 的引脚兼容,所以可以把 2764 直接插在原 6264 的插座上。这样,程序就不会由于断电而丢失。

2764 各引脚的定义如下:

$A_0 \sim A_{12}$: 13 根地址输入线。用于寻址片内的 8K 个存储单元。

$D_0 \sim D_7$: 8 根双向数据线。正常工作时为数据输出线;编程时为数据输入线。

\overline{CE} : 选片信号。低电平有效,当 $\overline{CE} = 0$ 时表示选中此芯片。

\overline{OE} : 输出允许信号。低电平有效,当 $\overline{OE} = 0$ 时,芯片中的数据可由 $D_0 \sim D_7$ 端输出。

\overline{PGM} : 编程脉冲输入端。对 EPROM 编程时,在该端加上编程脉冲。读操作时 $\overline{PGM} = 1$ 。

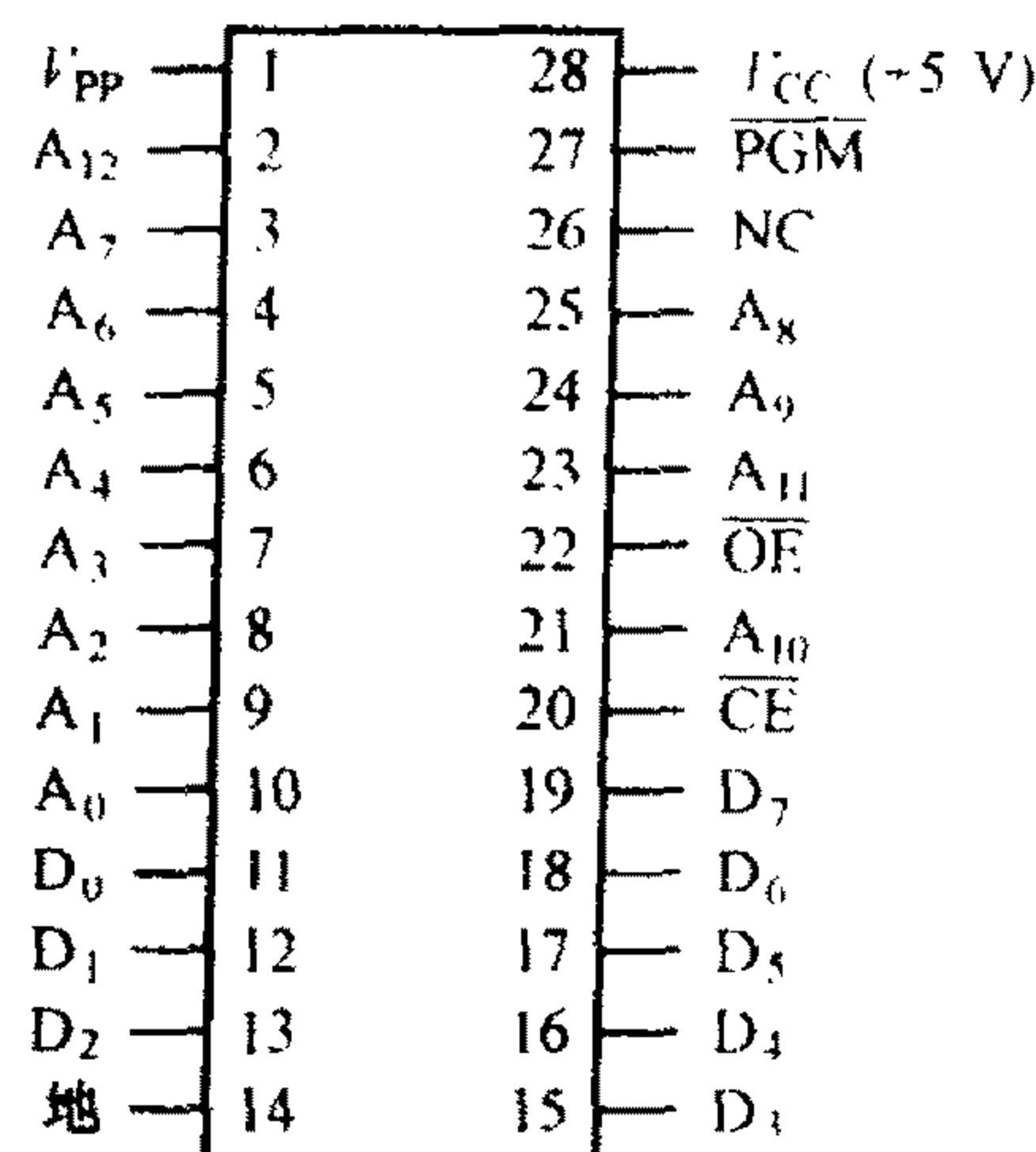


图 6.25 EPROM 2764 引脚图

V_{PP} : 编程电压输入端。编程时应在该端加上编程高电压,不同的芯片对 V_{PP} 的值要求不一样,可以是 +12.5 V、+15 V、+21 V、+25 V 等。

2) 2764 的工作方式

2764 可以工作在读出、编程写入、校验等七种方式下。表 6-2 给出了这七种方式的引脚电平设置。其中编程、编程禁止和校验属于写入器环境。即要将该 EPROM 芯片置入专门的写入器之中,并将有关的各引脚设为相应的状态,使其按地址顺序逐字写入,之后进行验证或令其处于禁止写入状态。如果要擦除信息,可用 12 mW/cm^2 功率的紫外线灯对芯片上方的石英玻璃窗口照射 10~20 分钟,则芯片的每一个编址单元被擦除为“FF”。

表 6-2 2764A 工作方式选择表

引脚 方式	$\overline{\text{CE}}$	$\overline{\text{OE}}$	$\overline{\text{PGM}}$	V_{PP}	V_{CC}	$D_0 \sim D_7$
读	低	低	高	V_{CC}	5 V	数据输出
输出禁止	低	高	高	V_{CC}	5 V	高阻
备用	高	x	x	V_{CC}	5 V	高阻
编程	低	高	低	12.5 V	5 V	数据输入
校验	低	低	高	12.5 V	5 V	数据输出
编程禁止	高	x	x	12.5 V	5 V	高阻
标识符	低	低	高	V_{CC}	5 V	制造商编码器件编码

2764 通常使用的方式是读方式,此时两个电源引脚 V_{CC} 和 V_{PP} 均连接 +5 V,编程脉冲输入端 $\overline{\text{PGM}}$ 接高电平,由于 2764 与 6264 SRAM 在引脚上是兼容的,所以其余的引线端在与系统的连接使用上都可按与 RAM 芯片相同的方法来进行电路设计。图 6.26 所示是 2764 芯

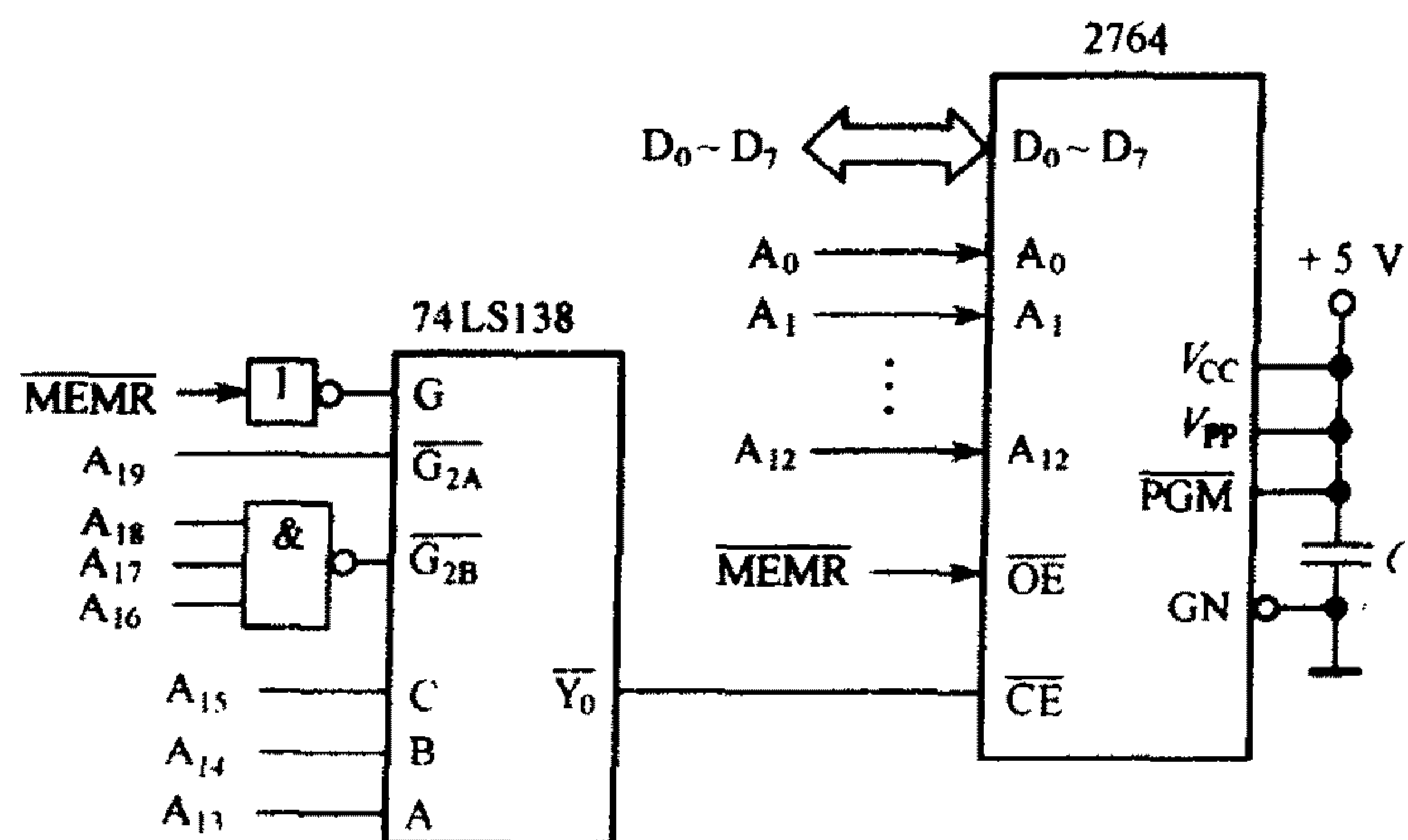


图 6.26 EPROM 2764 与 8088 系统的连接图

片与 8088 总线的连接图。由图可以看出,2764 芯片的地址范围为 70000H ~ 71FFFH。

6.3.4 电擦除可重写只读存储器 (EEPROM 或 E²PROM)

EEPROM 是“电擦除可编程只读存储器”的英文缩写。由于采用加高电压擦除的技术,所以它允许在线编程写入和擦除,而不必像 EPROM 那样需要从线路板上取下来。从这一点讲,它的使用要比 EPROM 方便。另外,EPROM 虽可多次编程写入,但整个芯片只要有一位写错,就必须从电路板上取下来全部擦掉重写,这给实际使用带来很大不便。因为在实际使用中,多数情况下需要的是以字节为单位的擦除和重写,而 EEPROM 在这方面就具有了很大的优越性。

与 EPROM 构造不同,EEPROM 在浮栅上又增加了一个控制栅,见图 6.27。擦除时将控制栅接地,同时在 S 极加较高的正电压,置浮栅于较强的电场之中,在电场力的作用下,浮栅上的自由电子会越过绝缘层进入源极,达到擦除的目的。

EEPROM 既可以实现正常工作方式中的只读不写,又可以将整个芯片或某个指定单元擦除。对指定的单元产生电流而使其他未通过电流的单元维持不变,从这个特点看,它具有了 RAM 的随机读/写能力。因此

EEPROM 允许选用两种擦除方式:字擦除方式,有选择地擦出一个字单元;数据块擦除方式,对某个数据块中的单元全部擦除。EEPROM 擦除一个单元比 EPROM 要快得多,但与 RAM 的写入速度相比仍相差甚远。

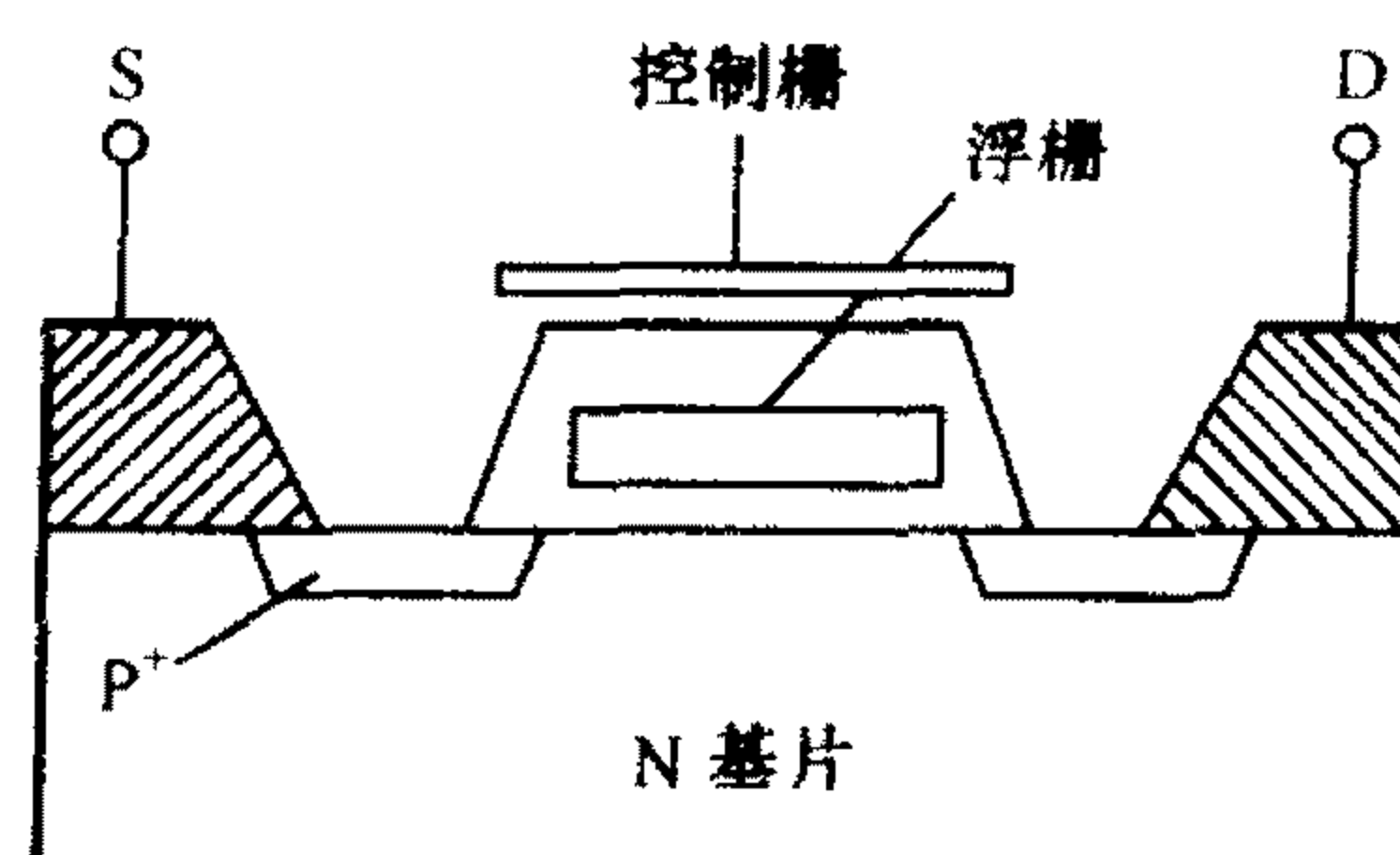


图 6.27 EEPROM 基本结构

6.3.5 闪速存储器 (Flash Memory)

理想的存储器应具有密度高、读/写快、成本低、和不挥发(不易失)的特点,而传统的存储器却只能满足这些要求中的一部分。闪速存储器(Flash Memory),又称快擦写型存储器能够实现 RAM 与 ROM 的所有功能。与传统存储器相比,Flash 的主要优点是:

① 不挥发性 相比 SRAM,Flash 不需后备电源来保持信息。

② 易更新性 相对于 EPROM 的紫外线擦除工艺,Flash 的电擦除功能为开发者节省了时间,也为用户更新存储器内容提供了可能。而与 EEPROM 相比,Flash 的成本更低,密度和可靠性更高。

由于这种芯片在断电后仍能长久保持信息,所以在现阶段,它除了取代 EPROM 和 EEPROM 来存放主板和显卡的 BIOS(基本输入/输出系统)以外,Flash Memory 还广泛应用于

便携式计算机的 PC 卡存储器(固态硬盘)。Flash Memory 芯片耗电低、集成度高、体积小、可靠性高。且读取速度大大高于硬盘驱动器,再加上没有机械运动部件,抗震性能好,非常适合于像便携机这样的移动设备。

以往主板所使用的 BIOS 在出厂时一次写入,不能更改,而更换 ROM 芯片对于普通用户难于实现,给 BIOS 程序的升级带来极大不便。利用 Flash Memory 存储主板的 BIOS 程序,实现升级则易如反掌。

一些微控制器内部也引入了片内 Flash 技术,集成片内 Flash 的 8 位微控制器,具有应用方便,工作可靠的特点。以 Motorola 微控制器为例,主要体现在:

① 单一电源电压供应 Flash 在正常的只读情况下,只需要为其提供普通的工作电压(+5 V)。如果用户要对 Flash 编程,需要同时提供高出正常工作电压的编程电压(如 +9 V)。Motorola 微控制器通过在片内集成电荷泵,便可以在片内产生编程电压。这使得用户无需为 Flash 的编程再增加额外的电源。

② 可靠性高 Motorola 片内 Flash 的存储数据可以保持 10 年以上,可擦写次数也在 1 万次以上。

③ 擦写速度快 Motorola 片内 Flash 的整体擦除时间可以控制在 5 ms 以内,对单字节的编程时间也在 40 μ s 以内。

④ Motorola 片内 Flash 支持在线编程,允许为控制器内部运行的程序去改写 Flash 存储内容。这一技术大大增加了 Motorola 微控制器的应用范围和使用方便性。

根据工艺,Flash 主要有两类: NAND Flash 和 NOR Flash。NOR Flash 是在 EEPROM 的基础上发展起来的,它的存储单元由 NMOS 构成,而连接 NMOS 单元的线都是独立的。NOR Flash 的特点是可以随机读取任意单元的内容,适合于程序代码的并行读/写存储,所以常用于制作 PC 机的 BIOS 存储器和微控制器内部存储器等。NAND Flash 将几个 NMOS 单元用同一根线连接,可以按顺序读取存储单元的内容,适合于数据或文件的串行读/写存储。

下面以 TMS28F040 芯片为例简单介绍闪存的工作原理。

1. 28F040 的引脚及结构

28F040 的外部引脚如图 6.28 所示。它共有 19 根地址线和 8 根数据线,说明该芯片的容量为 512 K \times 8 bit; \bar{G} 为输出允许信号,低电平有效; \bar{E} 是芯片写允许信号,在它的下降沿锁存选中单元的地址,用上升沿锁存写入的数据。

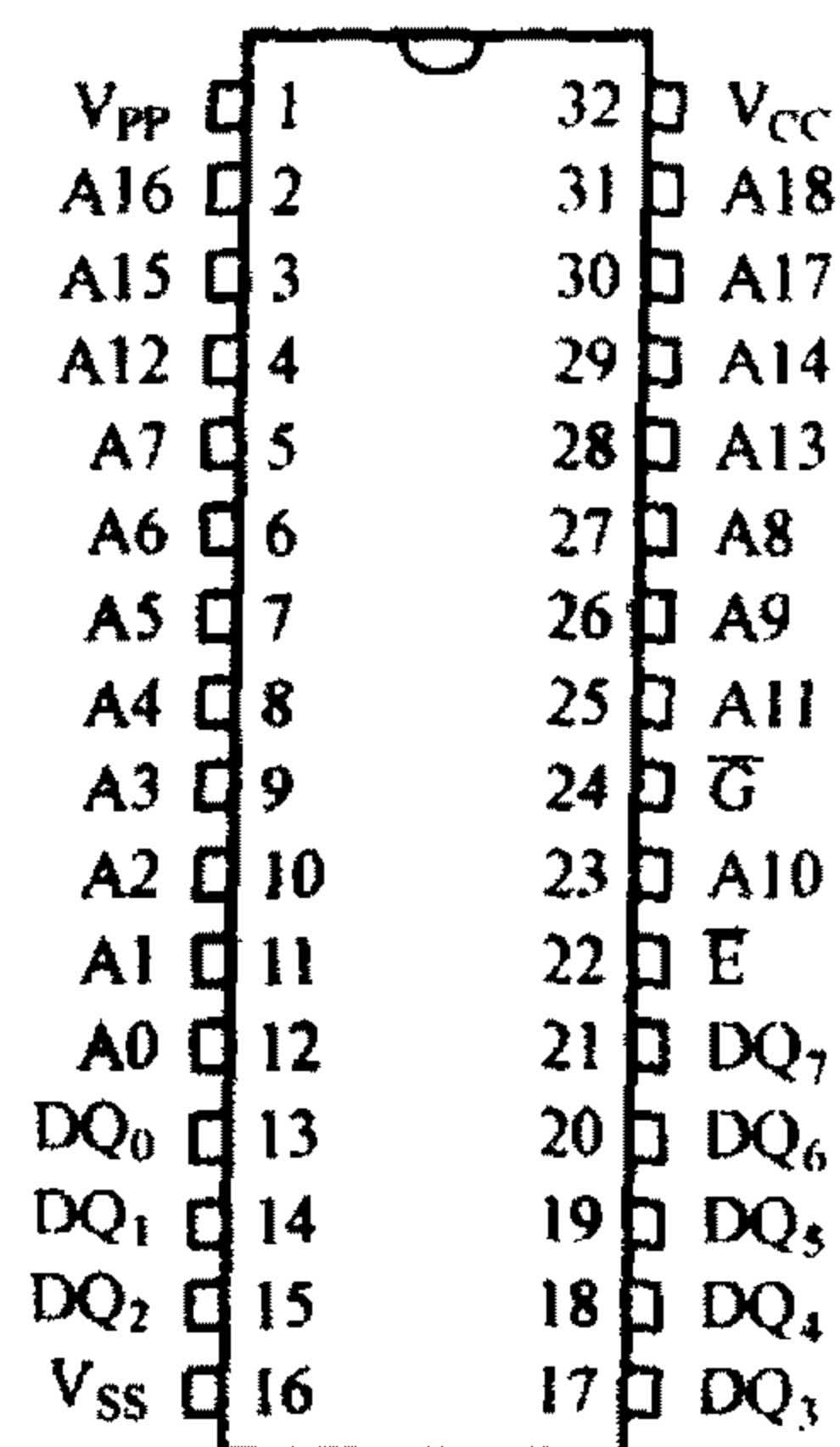


图 6.28 28F040 的引脚图

28F040 芯片将其 512 KB 的容量分成 16 个 32 KB 的块(或页),每一块均可独立进行擦除。

2. 工作过程

28F040 也有读出、编程写入和擦除三种工作方式。但不同的是它是通过向内部状态寄存器写入命令的方法来控制芯片的工作方式,对芯片所有的操作都要先向状态寄存器写入命令。另外,28F040 的许多功能需要根据状态寄存器的状态来决定。要取得芯片当前的工作状态,只需写入命令 70H,就可读出状态寄存器各位的状态了。状态寄存器各位的含义和 28F040 的命令分别见表 6-3 和表 6-4。

表 6-3 状态寄存器各位的含义

位	高电平(1)	低电平(0)	用于
SR ₇ (D ₇)	准备好	忙	写命令
SR ₆ (D ₆)	擦除挂起	正在擦除/已完成	擦除挂起
SR ₅ (D ₅)	块或片擦除错误	片或块擦除成功	擦除
SR ₄ (D ₄)	字节编程错误	字节编程成功	编程状态
SR ₃ (D ₃)	V _{PP} 太低,操作失败	V _{PP} 合适	监测 V _{PP}
SR ₂ ~ SR ₀			保留未用

28F040 对芯片的编程写入采用字节编程方式,其写入过程如图 6.28 所示。

首先,28F040 向状态寄存器写入命令 10H,再在指定的地址单元写入相应数据。接着查询状态,判断这个字节是否写好。未写好则重复这个过程,直到全部字节写入完毕。28F040 对一个字节的写入时间只需 8.6 μs。

1) 读操作

读操作包括读出芯片中某个单元的内容、读内部状态寄存器的内容以及读出芯片内部的厂家及器件标记三种情况。如果要读某个存储单元的内容,则在初始加电以后或在写入命令 00H(或 FFH)之后,芯片就处于只读存储单元的状态。这时就和读 SRAM 或 EPROM 芯片一样,很容易读出指定的地址单元中的数据。此时的 V_{PP}(编程高电压端)可与 V_{CC}(+5 V)相连接。

表 6-4 28F040 的命令字

命 令	总线周期	第一个总线周期			第二个总线周期		
		操作	地 址	数 据	操作	地 址	数 据
读存储单元	1	写	×	00H			
读存储单元	1	写	×	FFH			
读标记	3	写	×	90H	读	IA①	
读状态寄存器	2	写	×	70H	读	×	SRD④
清除状态寄存器	1	写	×	50H			
自动块擦除	2	写	×	20H	写	BA②	D0H
擦除挂起	1	写	×	B0H			
擦除恢复	1	写	×	D0H			
自动字节编程	2	写	×	10H	写	PA③	PD⑤
自动片擦除	2	写	×	30H	写		30H
软件保护	2	写		0FH	写	BA②	PC⑥

注:其中:① 若是读厂家标记,IA = 00000H;读器件标记则 IA = 00001H;

② BA 为要擦除块的地址;

③ PA 为欲编程存储单元的地址;

④ SRD 是由状态寄存器读出的数据;

⑤ PD 为要写入 PA 单元的数据;

⑥ PC 为保护命令,若 PC = 00H—清除所有的保护,PC = FFH—置全片保护,PC = F0H—清地址指定的块保护,PC = 0FH—置地址指定的块保护。

2) 编程写入

编程方式包括对芯片单元的写入和对其内部每个 32 KB 块的软件保护。软件保护是用命令使芯片的某一块或某些块规定为写保护,也可置整片为写保护状态,这样可以使被保护的块不被写入新的内容或擦除。如向状态寄存器写入命令 0FH,再送上要保护块的地址,就可置规定的块为写保护。若写入命令 FFH,就置全片为写保护状态。其写入过程如图 6.29 所示。

3) 擦除方式

28F040 既可以每次擦除一个字节,也可以一次擦除整个芯片,或根据需要只擦除片内某些块,并可在擦除过程中使擦除挂起和恢复擦除。

对字节的擦除不难理解,写入数据的同时就等于擦除了原单元的内容。对整片擦除,其标志是擦除后各单元的内容均为 FFH。整片擦除最快只需 2.6 s。但受保护的内容不被擦除。也允许对 28F040 的某一块或某些块擦除,每 32 KB 为一块,块地址由 A₁₅ ~ A₁₈ 来决定。

在擦除时,只要给出该块的任意一个地址(实际上只关心 $A_{15} \sim A_{18}$)即可。整片擦除及块擦除的流程图分别如图 6.30 中的(a)和(b)所示。擦除一块的最短时间为 100 ms。

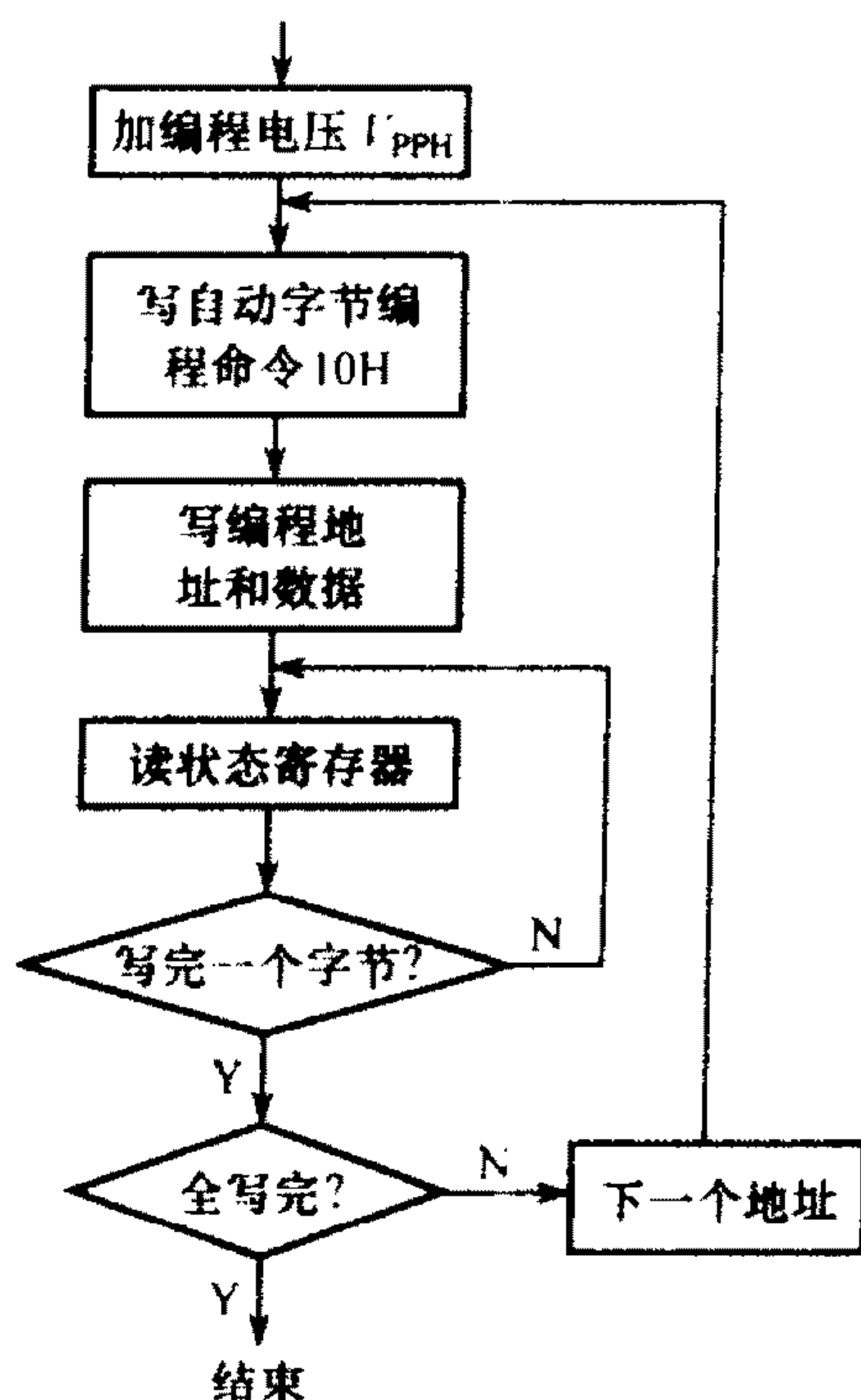


图 6.29 28F040 的字节写入过程

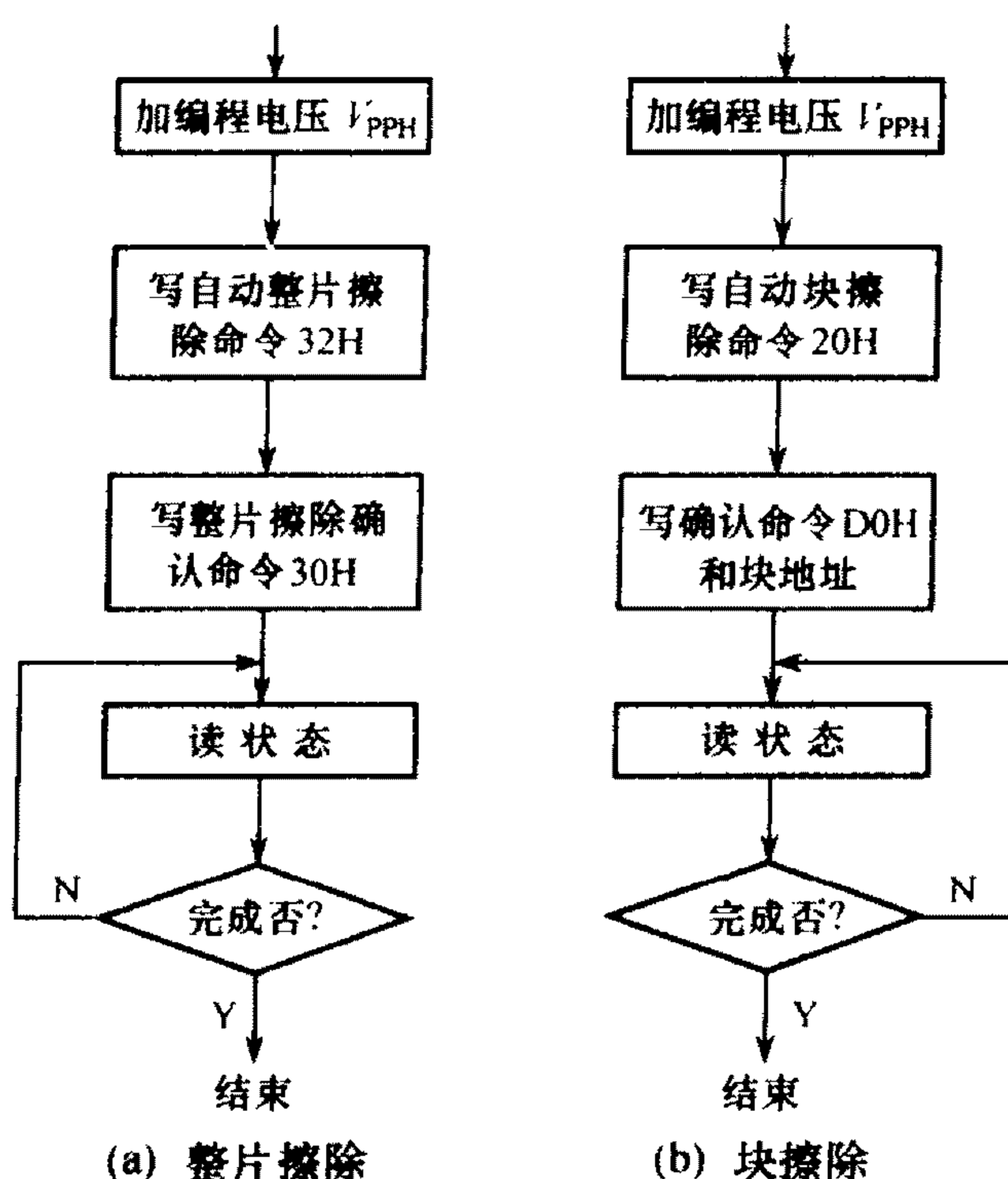


图 6.30 28F040 的擦除流程

擦除挂起是指在擦除过程中需要读数据时,可以利用命令暂时挂起擦除,读完后又可用命令恢复擦除。

28F040 在使用中,要求在其引脚控制端加上适当电平,以保证芯片正常工作。不同工作类型的 28F040 的工作条件是不一样的,具体见表 6-5。

表 6-5 28F040 的工作条件

	E	G	V_{PP}	A_9	A_0	$D_0 \sim D_9$
只读存储单元	V_{IL}	V_{IL}	V_{PPL}	x	x	数据输出
读	V_{IL}	V_{IL}	x	x	x	数据输出
禁止输出	V_{IL}	V_{IH}	V_{PPL}	x	x	高阻
准备状态	V_{IH}	x	x	x	x	高阻
厂家标记	V_{IL}	V_{IL}	x	V_{ID}	V_{IL}	97H
芯片标记	V_{IL}	V_{IL}	x	V_{ID}	V_{IH}	79H
写入	V_{IL}	V_{IH}	V_{PPH}	x	x	数据写入

注:表中 V_{IL} 为低电平; V_{IH} 为高电平 V_{CC} ; V_{PPL} 为 $0 \sim V_{CC}$; V_{PPH} 为 +12 V; V_{ID} 为 +12 V; x 表示高、低电平均可。

附带介绍一点,由 28F040 等 28、29 系列的 Flash ROM 芯片制成的 BIOS,其明显的特点就是升级方便,目前国内比较流行的主板 BIOS 主要有 Award BIOS、AMI BIOS, Award BIOS 是由 Award Software 公司开发的 BIOS 产品,功能较为齐全,在目前的主板中使用最为广泛。在 Award 和 AMI 等公司的网站中就提供与主板厂家配套的 BIOS 升级文件。如: <http://www.award.com.tw/download/bios/>。但用户在下载的时候,必须先清楚自己主板 BIOS 的型号,主板芯片组的型号,最新版本的 BIOS 增加了哪些功能,对自己的机器有无实用价值等。升级 BIOS 必须慎重行事,关于升级 BIOS 的具体操作,有兴趣的读者可以阅读自己主板的用户手册和其他有关资料。

6.4 微型计算机系统存储器的组织

以上介绍的半导体存储器芯片(SRAM、DRAM、ROM)是组成主存储器的器件。以下先讨论如何用存储芯片组成一个主存储器。

6.4.1 存储器的扩展技术

设计存储器时,首先确定所需要的总容量,即字数(位数)。字数即可编址单元数,简称单元数。大多数主存都按字节编址,也就是每个编址单元为 8 位。

在微型计算机系统中,由单个芯片不可能构成主存储器,设计者需要根据存储器总容量和所选用的存储芯片的型号、容量来决定芯片的数量,通常叫做字扩展和位扩展。

1. 位扩展

如果选用的存储芯片位数小于存储系统编址单元的位数,则需要进行位数扩充。假定某台微型计算机的主存容量为 $1\text{M} \times 8$ 位,即通常意义的 1 MB,可采用 8 片 $1\text{M} \times 1$ 或 2 片 1×4 位的存储芯片连接。

位扩展的连接方式是将多片存储器的地址、片选、读/写输入引线端相并联,数据端单独引出。如图 6.31 所示的位扩展方式是由 $16\text{K} \times 1$ bit 芯片扩展为 $16\text{K} \times 8$ bit 的存储器。图中每个芯片字长 1 位,存储器字长 8 位,因此它由 8 片芯片并联组成。每片有 14 条地址线,每根地址线接有 8 个芯片;每片有一根数据线,每根数据线接有一个芯片。

2. 字扩展

字扩展适于位数不变,而字数(单元数)不够,需要若干芯片组成满足总容量的存储器。为此将高位地址译码产生若干不同的片选信号,按各芯片在存储空间分配中所占的地址范围,分送各芯片(参见 6.2.2 中图 6.12 的 SRAM 6116 与系统的连接);低位地址线直接连接

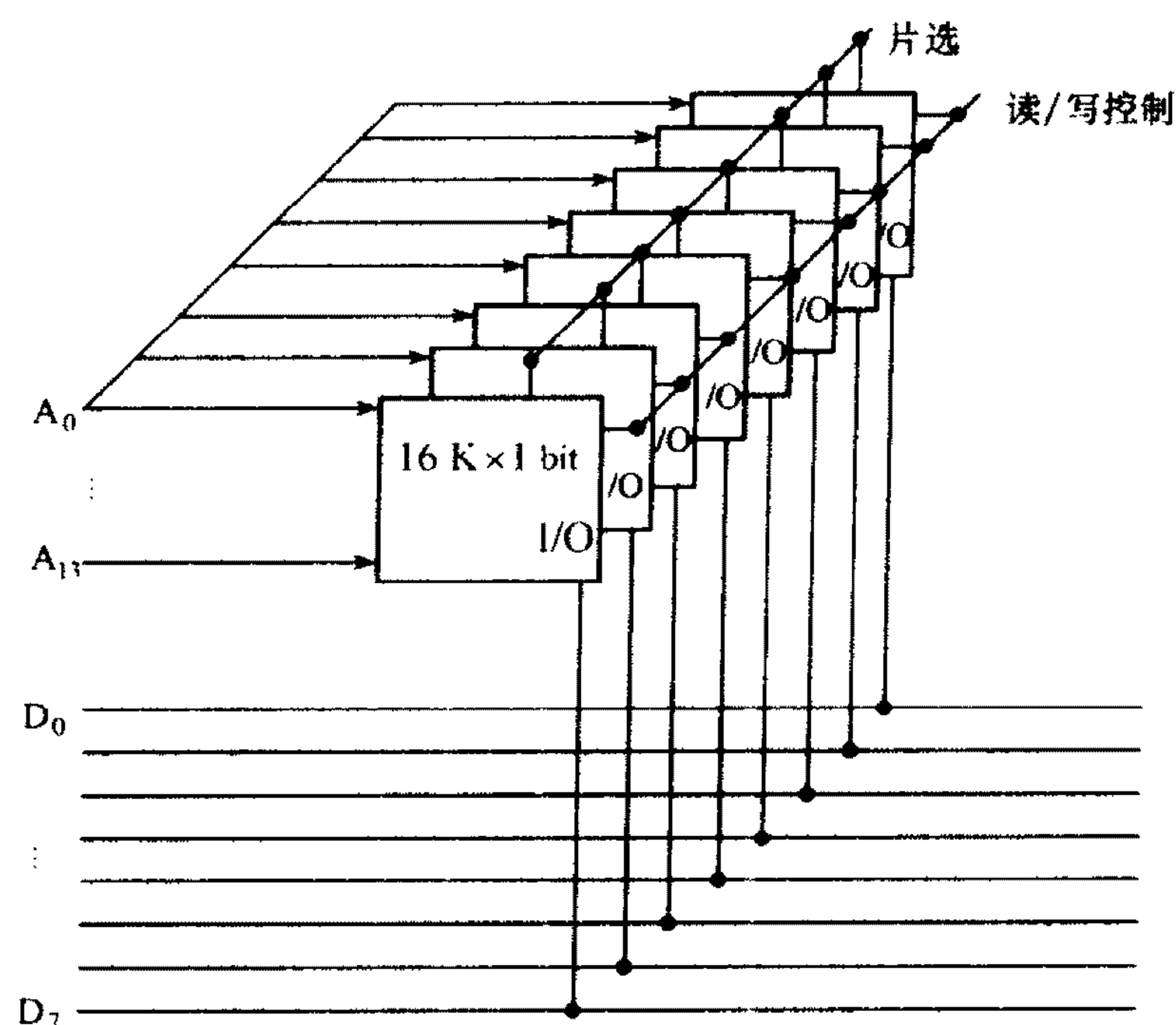


图 6.31 位扩展连接方式

各芯片的地址线,以选择片内的某个单元,而各芯片的数据线则按位与数据总线并联。位扩展连接方式如图 6.31 所示。当需要访问某个内存单元,则向存储器送出地址码,使得一个芯片的片选信号有效,而低位地址在该芯片内部选中某个单元。如图 6.32 所示字扩展方式是由 $16\text{ K} \times 8\text{ bit}$ 芯片扩展成 $64\text{ K} \times 8\text{ bit}$ 存储器。图中为 4 个芯片并联,数据总线 $D_0 \sim D_7$ 与各片的数据线相连,低位地址总线 $A_0 \sim A_{13}$ 与各片的 14 位地址线相连,两位高位地址 A_{14} 、 A_{15} 经过译码器和 4 个片选端相连。这 4 片的地址分配见表 6-6。

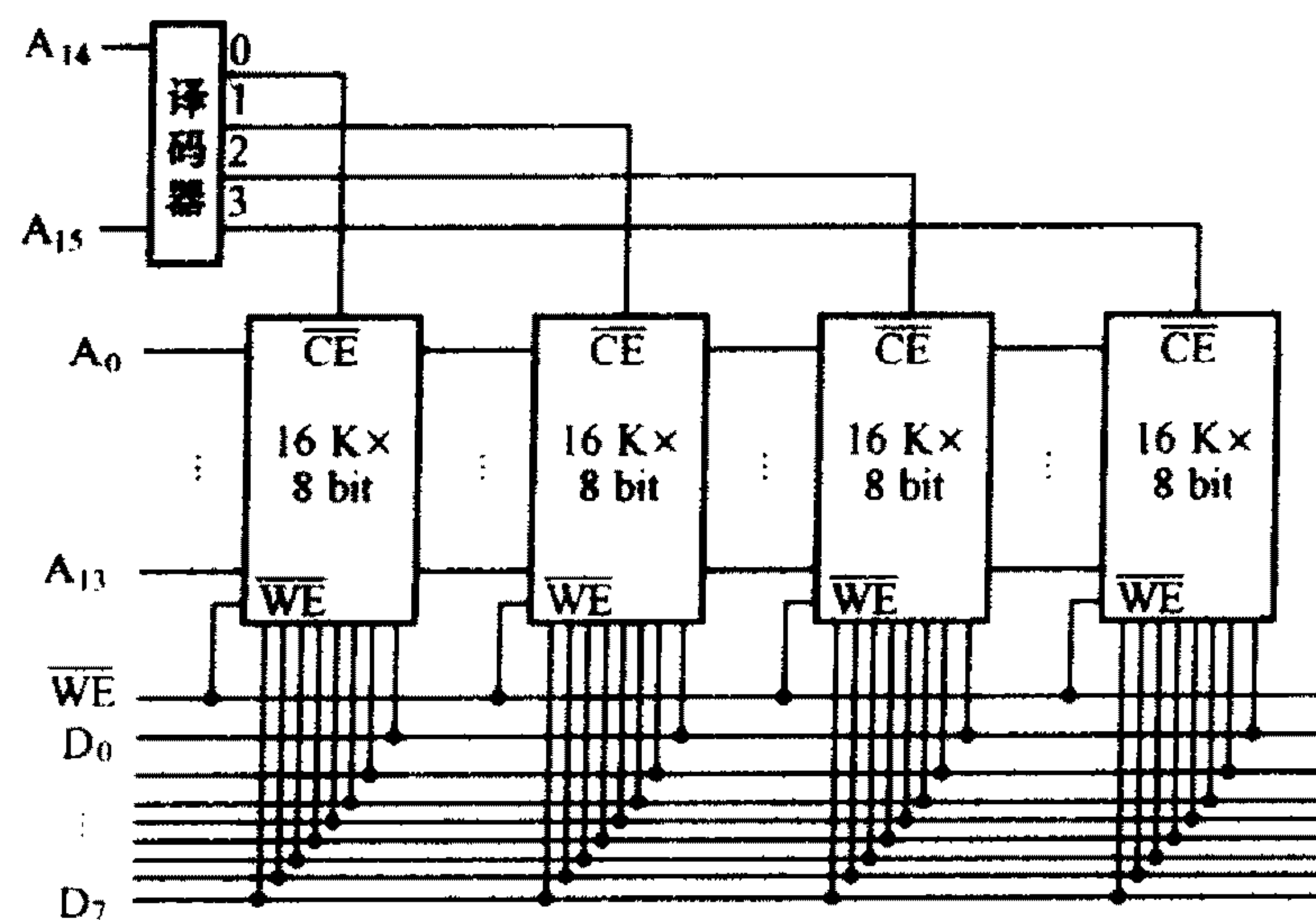


图 6.32 字扩展连接方式

表 6-6 芯片选取与存储空间分配

片号		A ₁₅	A ₁₄	A ₁₃ …A ₀	十六进制范围
0	低地址	0	0	00000000000000	0000
	高地址	0	0	11111111111111	3FFF
1	低地址	0	1	00000000000000	4000
	高地址	0	1	11111111111111	7FFF
2	低地址	1	0	00000000000000	8000
	高低址	1	0	11111111111111	BFFF
3	低地址	1	1	00000000000000	C000
	高低址	1	1	11111111111111	FFFF

3. 字位扩展

若存储器总容量为 $M \times N$ 位, M 为存储器的单元数 $M = 2^L$, N 为每个单元的二进制位数, 而现有存储芯片为 $m \times n$ 位, 且 $M > m, N > n$, 则需要字位同时扩展, 这个存储器所需要的芯片数为 $(M/m) \times (N/n)$ 。

6.4.2 CPU 与主存储器的连接

主存与 CPU 是两个独立的部件, 如何进行连接, 从原理上考虑, 大体上有以下几个方面:

1) 系统模式

① 最小系统模式 图 6.33 所示是一个小容量存储器与 CPU 的直接连接, 即 CPU 的地址线、数据线直接送往存储芯片, 由于存储器容量不大, 往往采用 SRAM 芯片, 以简化逻辑。图中存储器由 2114 芯片经字、位扩展构成, 容量为 $4\text{ K} \times 8\text{ bit}$ 。2114 芯片本身只有 $1\text{ K} \times 4\text{ bit}$, 整个存储器共需要 $(4\text{ K}/1\text{ K}) \times (8/4) = 8$ 片。2114 有 10 个地址信号端 $A_0 \sim A_9$ 、4 位数据信号端 $D_0 \sim D_3$ (或 $D_4 \sim D_7$)、一个片选信号 $\overline{\text{CE}}$ 和一个读/写信号 $\overline{\text{WE}}$ 。由 CPU 提供 12 位地址, 低 10 位 $A_0 \sim A_9$ 并行连接各芯片的地址端, 两位高地址 A_{10} 、 A_{11} 连接译码器, 以产生 4 个片选信号, 分别控制 4 组芯片。此外, 译码器还要受 CPU 的访存信号 $\overline{\text{MREQ}}$ 控制, 只有需要访问主存时才产生译码输出。CPU 的读/写信号 $\overline{\text{WE}}$ 控制对存储器的读/写操作。

② 较大系统模式 在现阶段, 大多数微型计算机系统都是通过系统总线连接主存储器和外围设备。前面章节提到, 一般 CPU 芯片的引脚不直接和系统总线相连, 而是通过数据收发缓冲器、地址锁存器、总线控制器等接口芯片与系统总线相连, 如图 6.34 所示。而主存储器模块就挂接在系统总线上, 通过总线完成一次存储器读/写, 需占用一个总线周期。

③ 专用存储总线模式 高档微型计算机要求访问内存速度较高, 则在 CPU 与主存储器

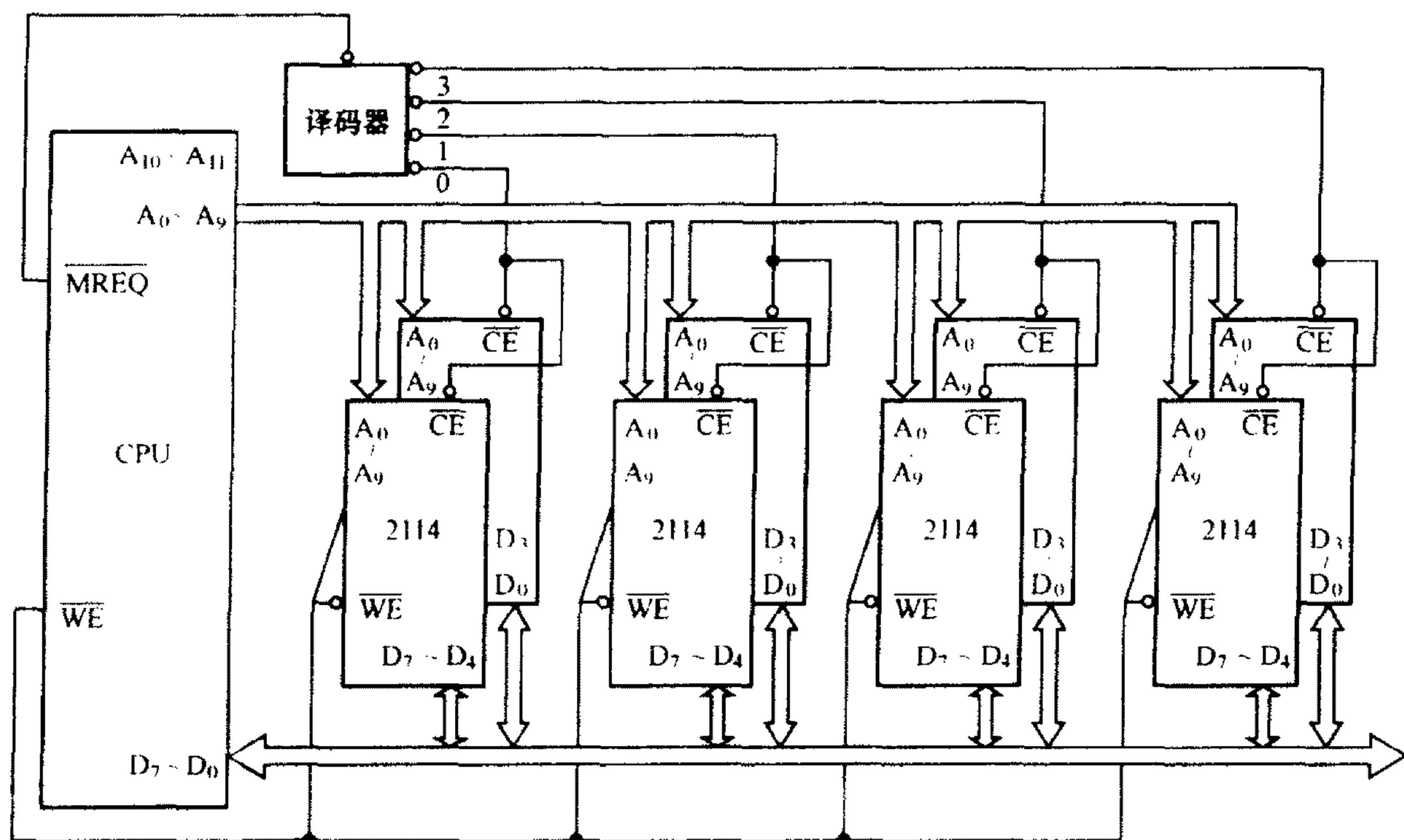


图 6.33 CPU 和存储器的直接连接

之间,专门设置了一组高速存储总线,其中包括地址线与数据线及少量控制信号。在这种系统模式中,CPU 通过存储总线访问主存储器,通过 I/O 总线访问 I/O 设备。

2) CPU 与主存速度的协调

如何协调 CPU 与主存储器之间工作速度的差异,通常有两种方式可供选择。

① 同步方式 这种方式是以存储器的存取周期来确定 CPU 的工作周期,使 CPU 内部操作和访存操作置于相同的时钟周期,但由于 CPU 的速度往往高于主存速度,这种方式使 CPU 效率降低。

② 准同步方式 在这种方式中,CPU 的工作周期与存储器的存取周期没有直接关系,二者之间通过一套信息交换规约建立联系。通常是当 CPU 进行一次内存的读/写时,若内存读/写周期未结束,CPU 以自身时钟周期为单位插入等待时钟,当存储器完成操作时发出一个就绪信号 READY。这种方式以时钟周期为单位与内存同步。故称为准同步方式,准同步方式较好地发挥了 CPU 的效率,通信规约也比较简单,为目前大多数计算机系统所采用。

3) 主存的控制信号

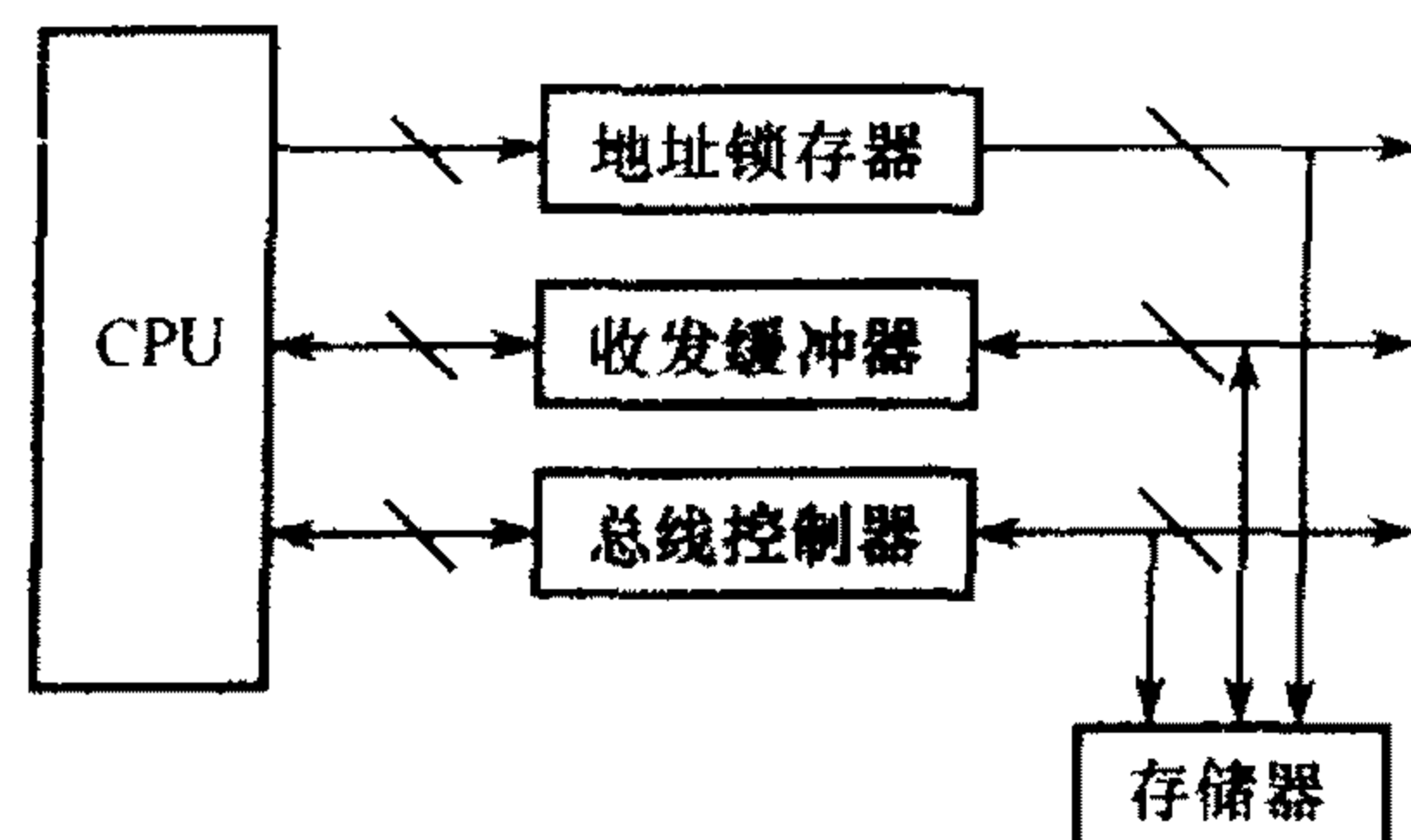


图 6.34 较大系统连接方式

前面已提到,存储芯片本身只需要最基本的控制命令,如 R/\overline{W} 、 \overline{CS} ,或将 \overline{CS} 分解为 \overline{RAS} 和 \overline{CAS} 。但为了实现对存储器的选择、容量的扩展、速度匹配,系统总线可能引伸出一些控制与应答信号。如 8088 总线提供了 M/\overline{IO} ,低电平时选中主存,高电平时选中输入/输出设备。有的系统总线将选择命令与读/写命令结合起来,分为两个控制信号: \overline{MEMW} 为存储器写, \overline{MEMR} 为存储器读。它们将参与控制片选的产生,并形成存储芯片所需的 R/\overline{W} (\overline{WE})或 \overline{RD} 。

6.4.3 PC 机的存储器组织

1. 存储空间的分配

以结构相对简单的 8088PC 机为例,8088 系统总线有 20 根地址线,可寻址的物理空间为 1 MB,地址编号为 0 ~ FFFFFH。这 1 MB 的存储空间被划分为三个区域:把前 640 KB 的内存作为主存 RAM;其后的 384 KB 分为 256 KB 的 ROM 空间和为 I/O 通道保留的 128 KB RAM 空间(保留区)。8088 微型计算机存储器空间的分配见表 6-7。

表 6-7 PC/XT(8088)存储空间分配表

地址范围	区域	功能
0 ~ 7FFFFH	系统板上 512 KB	系统板存储器
80000H ~ 9FFFFH	128 KB 基本 RAM	I/O 通道主存储器
A0000H ~ BFFFFH	128 KB 显示 RAM	保留给显示卡
C0000H ~ EFFFFH	192 KB 控制 ROM	保留给硬盘适配器、显卡
F0000H ~ FFFFFH	系统板上 64 KB ROM	系统 BIOS 用

在 8088 时代,PC 机的应用程序规模小,用户界面为字符行形式,无需处理发展到今天的多媒体信息,按当时计算机发展的水平,人们都认为 640 KB 的存储容量对一般应用已经够了,并没有人对 1 MB 的内存空间提出什么异议。时至今日,PC 机的地址总线宽度由 20 位、24 位、32 位发展到 Pentium 系统中的 36 位,相应的可寻址内存空间由 1 MB、16 MB (80286)、4 GB(80386)到 64 GB(Pentium)。主板系统支持的物理存储体总量也由 1 MB、16 MB、32 MB、512 MB 到 2 GB。

为保持兼容性,MSDOS 把地址范围 0 ~ 9FFFFH 的 640 KB 的主存空间称为基本内存或常规内存,把 A0000H ~ FFFFFH 的 384 KB 作为内存保留区,留做显示适配器和 ROM BIOS 使用。对于 PC/XT 机,这 384 KB 存储空间分别附在各个接口卡上,如 VGA 卡等,并不能用来存取数据,故称为“保留区”。而把地址在 100000H 以上的存储器称为扩展存储器(Extended Memory),也称 XMS。在这之上的主存储空间称为扩展内存。

由于当时的 ROM BIOS 和视频存储区并未占满 384 KB,有一部分空闲区,被称为上位内

存块,简称 UMB,其大小因系统所配置的显示适配器和接口卡的不同而改变。DOS 系统使用专门的软件来管理这部分内存。

对于 1 MB 以上的扩展内存,DOS 系统显然不能直接访问和管理,解决的方法是:从上述空闲的空间中划出 64 KB(通常分为 4 页,每页 16 KB)作为应用程序利用扩充内存的窗口。这 64 KB 的空间称为页框。1 MB 以上的存储器也按 16 KB 分页,每次可交换 4 页内容。通过 EMS 管理程序(扩充内存管理规范),可以把该页框中的 4 个页面与实际存在的扩充内存中的物理页面交换内容,每当程序要使用扩充存储器时,EMS 就会把扩充存储器中的页面拷贝到页框的区域内,然后程序就可以从该页框中来访问这些页面(如图 6.35 所示)。因此,通过内存保留区中页框的映射,就可访问全部 EMS 存储器,使 DOS 环境下用户拥有的内存远远超出了原来 640 KB 的限制。读者可能注意到“扩充”内存和“扩展”内存的说法,实际上 EMS 内存就是扩展内存 XMS,但它是利用扩充存储器的管理程序通过页框窗口来访问 XMS 存储器。也可以说一个指“存在”,一个指“实现”。

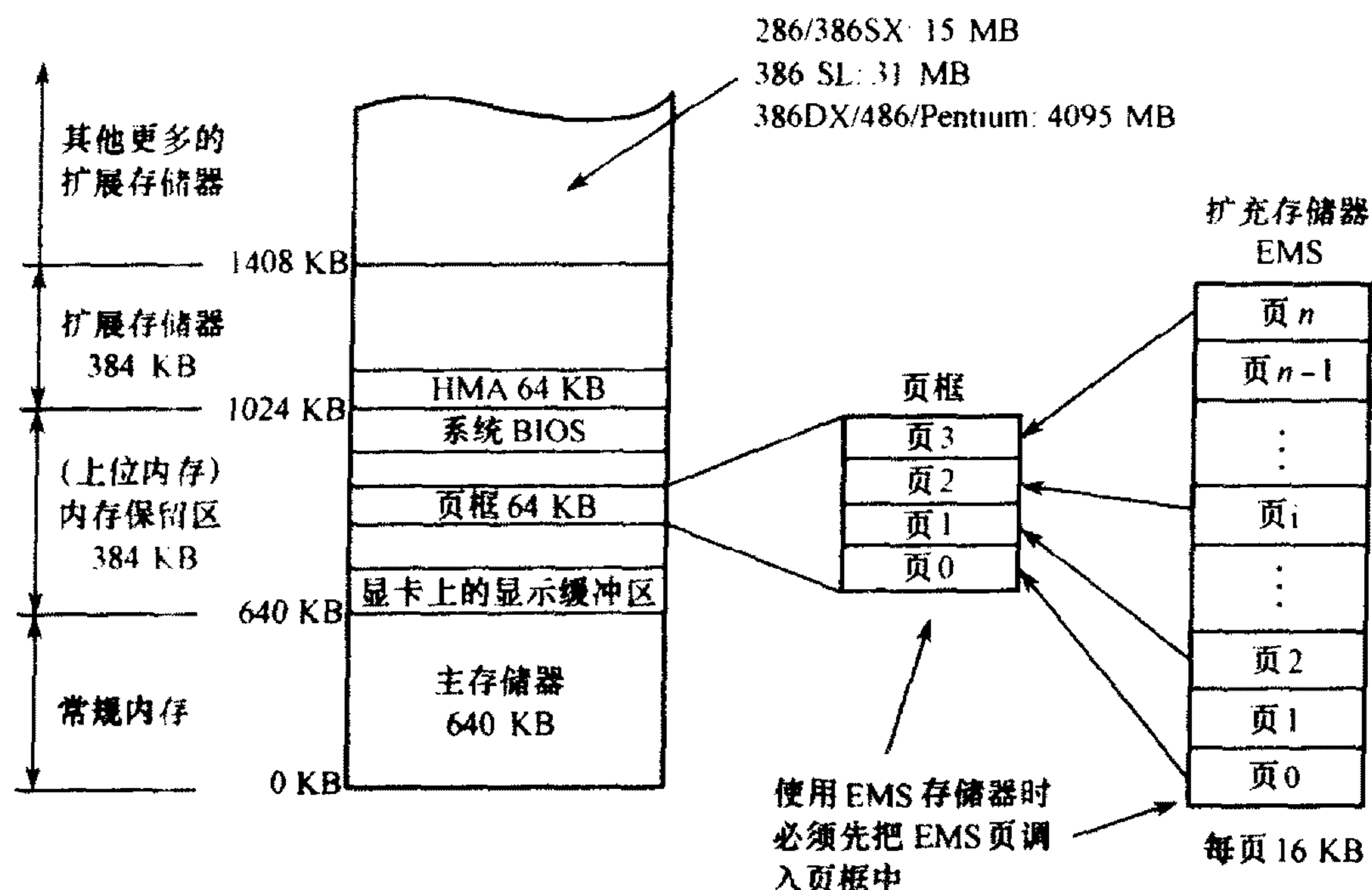


图 6.35 扩充存储器管理示意图

2. 存储器的实地址方式和保护工作方式

在 80386 以上档次的微型计算机中,Windows 操作系统逐步取代了 MSDOS。因为 PC 机的技术保持严格的连续性,仍把低地址范围的 640 KB 内存称为基本内存,在 640 KB 之上都称为扩展内存。对存储器的管理有实地址方式和保护方式(又叫做虚地址保护方式)两种工作方式。在实方式下,物理地址使用 20 位,所以最大寻址空间为 1 MB。同 8086/8088 的工

作方式是一样的,故又称 8086 工作方式。Windows 系统中的“MS-DOS 工作方式”即是提供了实地址方式下工作的入口。8086/8088 指令只能在实地址方式下运行,80386 的指令除了 9 条保护方式指令外,其余都可在实地址方式下运行。

保护方式采用 32 位物理地址,寻址范围可达 4GB(关于保护模式下的存储器管理见后面的内容)。另外,显卡及各接口卡的保留存储空间范围,由 Windows 系统根据所配置的系统内存的大小、各接口卡存储器的容量自动分配。

3. 几种 PC 机主板的存储器配置

① PC/XT 主板上的 RAM 安装 XT 机设有 4 组(或称体 Bank)RAM 插座,每组 9 片,包括 8 位数据片和 1 位奇偶校验位。通过主板的跳线和预制 DIP 开关状态,系统可识别主板上 RAM 芯片的型号和容量,主板上最大可安装 RAM 640 KB。增加扩展内存卡最大存储容量为 1 MB。

② M218 PC/AT 机主板 支持 DIP(双列直插)和 SIMM(单边接触直插式存储器模块)两种类型的 DRAM。设有 8 个 DIP 插座,用户可以用 8 片 $256\text{ K} \times 4$ 位 DIP DRAM 芯片构成 1 MB 的系统内存,也可以用 8 片 $1\text{ M} \times 4$ 位的 DIP DRAM 构成 4 MB 的系统内存。若使用 SIMM 内存条,可用 4 个 1 MB 或 4 个 4 MB 的 SIMM DRAM 内存条构成 4 MB~16 MB 的系统内存。系统支持的最大主存储容量为 16 MB。

随着存储器集成度日益提高和容量的增大,数据传输的密集度亦增大,而需要的数据传输线增多,DIP 的封装技术已不能满足要求。半导体技术的发展保证了存储器芯片容量增加而体积减小,从而产生了 SIMM 内存条(内存条的外形参见 6.8.1 节)。一般 1~4 MB 的 SIMM 有 30 条引脚,4~32 MB 的 SIMM 有 72 条引脚。内存的读/写方式采用快速页面模式。

③ ISA-486 主板 主板上有两个安装内存条的 Bank,即 Bank₀ 和 Bank₁,每一个 Bank 由 4 个 SIMM 插槽组成,每个 Bank 可以安装 1 MB、4 MB 或 16 MB 的内存,在主板上最多可安装 32 MB 的内存,若增加内存扩充卡,最大系统内存容量为 64 MB。且不必再用跳线的方式设置内存的大小,系统 BIOS 会自动识别安装的内存容量(即插即用)。SIMM 的读/写方式为快速页面方式。

80486CPU 内部包含 8 KB 的高速缓存 SRAM,系统板上还可以安插两种类型的高速缓存:Cache RAM(常称为二级高速缓存)和用来保存 Cache RAM 地址的 Tag SRAM,以便当 CPU 访问内存时,先在 Tag SRAM 中查找地址,如果找到,意味着需要的数据在 Cache RAM 命中。可以安装 8 片 $8\text{ K} \times 8$ 的 SRAM 构成 64 KB 的缓存(存取速度 25 ns),并在 Tag SRAM 插座中插入 $8\text{ K} \times 8$ 的芯片,存取速度为 25 ns;若选用 8 片 $32\text{ K} \times 8$ 的 SRAM 芯片构成 256 KB 的缓存,则需要在 Tag SRAM 插座中插入 $32\text{ K} \times 8$ 的芯片,芯片速度同为 25 ns。最大可配置 256 KB 的缓存。选择不同的二级缓存配置需要设置短路端子。

④ 联想主板 SX2EP(Pentium III 处理器) 主板提供 3 组 168 线 DIMM(双边接触直插式

存储模块),芯片类型为 SDRAM (同步式 DRAM)。可安装的 SDRAM 容量最大为 512 MB。存取速度为 6 ~ 10 ns,工作频率可达 133 MHz。Pentium III 处理器一般内嵌了 8 KB 的一级缓存,并与 256 KB 的二级缓存封装在一起

6.5 高速缓存(Cache)

6.5.1 Cache 的工作原理和基本结构

前面提到,速度是存储器的主要指标之一,而 CPU 与内存之间的速度差异是影响计算机系统速度的“瓶颈”。解决这一问题在于保持 CPU 的能力,提高主存的速度。用硬件技术提高存储芯片的存取速度是一个有效手段;而在慢速的 DRAM 和快速的 CPU 之间插入一速度较快、容量较小的 SRAM 起到缓冲作用,即 Cache 技术,如图 6.36 所示。使得速度和成本之间的矛盾得到较合理的解决,是提高主存速度的另一个思路。目前的微型计算机系统中一般均采用这种方法来提高存储系统的性能,使系统在成本增加不高的情况下,性能有较显著的提升。

以下简要介绍 Cache 的概念、原理、结构设计以及在微型计算机和 CPU 中的实现。

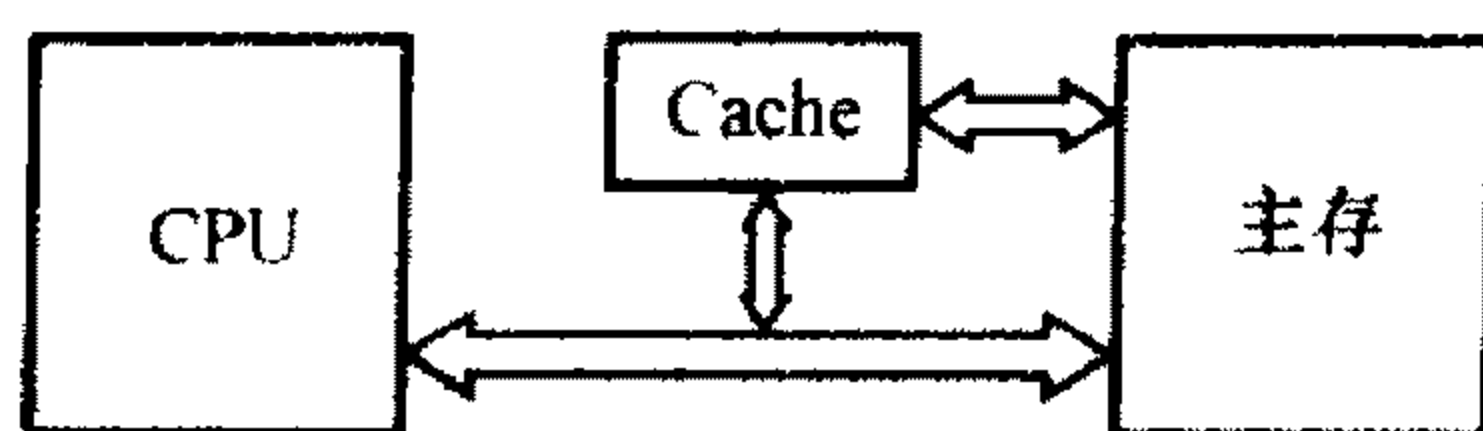


图 6.36 Cache 在微型计算机系统的位置

1. Cache 的工作原理

Cache 的工作原理是基于程序和数据访问的局部性。

任何程序或数据要为 CPU 所使用,必须先放到主存中。CPU 只与主存交换数据,所以主存的速度在很大程度上决定了系统的运行速度。对大量典型程序运行情况的分析结果表明,程序运行期间,在一个较短的时间间隔内,由程序产生的内存访问地址往往集中在存储器的一个很小范围的地址空间内。这一点不难理解。指令地址本来就是连续分布的,再加上循环程序段和子程序段要多次重复执行。因此,对这些地址中的内容的访问就自然具有时间上集中分布的倾向。数据分布的这种集中倾向不如指令明显,但对数组的存储和访问以及内存变量的安排都使存储器地址相对集中。这种在单位时间内对局部范围的存储器地

址频繁访问,而对此范围以外的地址则访问甚少的现象,被称为程序访问的局部化(Locality of Reference)性质或称为程序访问的局部性。

由此可以想到,如果把在一段时间内、一定地址范围内被频繁访问的信息集成批地从主存中读到一个能高速存取的小容量存储器中存放起来,供程序在这段时间内随时使用,从而减少或不再去访问速度较慢的主存,就可以加快程序的运行速度。这就是 Cache 的设计思想,在 CPU 和主存之间设置一个小容量的高速存储器,称为高速缓冲存储器,简称 Cache。可以看出,程序和数据访问的局部化性质是 Cache 得以实现的原理基础。

有了 Cache,系统在工作时,就总是不断地将与当前指令集相关联的一个不太大的后继指令集合从内存读到高速 Cache,然后再与 CPU 高速传送,从而达到速度匹配。在这种方案中,CPU 在读取指令或数据时,首先判别要求访问的字是否在 Cache 内,若找到则为“命中”;不在,则为“不命中”。如果“命中”,就直接对 Cache 相应的单元(由主存地址变换而来)进行高速读/写操作。

由于局部性原理不能保证所请求的数据百分之百地在 Cache 中,这里便存在一个命中率问题。所谓命中率,就是在 CPU 访问 Cache 时,所需信息恰好在 Cache 中的概率。命中率越高,正确获取数据的可能性就越大。如果高速缓存的命中率为 92%,可以理解为 CPU 在访问存储器时,用 92%的时间与 Cache 交换数据,8%的时间与主存交换数据。

一般来说,Cache 的存储容量比主存的容量小得多,但不能太小,太小会使命中率太低。但也没有必要过大,过大不仅会增加成本,而且当 Cache 容量超过一定值后,命中率随容量的增加将不会有明显的提高。所以,Cache 的空间与主存空间在一定范围内应保持适当比例的映射关系,以保证 Cache 有较高的命中率,并且系统成本不过大地增加。一般情况下,可以使 Cache 与内存的空间比为 1:128,即 256 KB 的 Cache 可映射 32 MB 内存;512 KB Cache 可映射 64 MB 内存。在这种情况下,命中率都在 90%以上。即 CPU 在运行程序的过程中,有 90%的指令和数据可以在 Cache 中取得,只有 10%需要访问主存。对没有命中的数据,CPU 只好直接从内存获取,获取的同时,也把它复制到 Cache 中,以备下次访问。

在有 Cache 的系统中,Cache 的命中率与 Cache 的大小、替换算法、程序特性等因素有关。增加 Cache 后,CPU 对主存的平均存取速度可按下式粗略地计算:

$$\text{系统平均存取速度} = \text{Cache 存取速度} \times \text{命中率} + \text{RAM 存取速度} \times (1 - \text{命中率}) \quad (6.5)$$

例 6-2 某微型计算机存储器系统由一级 Cache 和 RAM 组成。已知 RAM 的存取速度为 80 ns,Cache 的存取速度为 6 ns,Cache 的命中率为 85%,求该存储系统的平均存取速度。

解 由式(6.5)得

$$\text{系统的平均存取速度} = 6 \text{ ns} \times 85\% + 80 \text{ ns} \times 15\% = 5.1 \text{ ns} + 12 \text{ ns} = 17.1 \text{ ns}$$

可以看出,有了 Cache 以后,CPU 访问主存的速度大大提高了。但要注意的是,增加

Cache 只是加快了 CPU 访问存储器系统的速度,而 CPU 访问存储器系统仅是计算机全部操作的一部分,所以增加 Cache 对系统整体速度只能提高 10% ~ 20% 左右。另外,若访问 Cache 没有命中的话,CPU 还要访问主存,这时反而延长了存取时间。所以按式(6.5)计算出来的平均存取速度仅是一个粗略值。

2. 高速缓存的基本结构

这里是指 Cache 的地址映像结构,也就是说,把存放在主存中的程序按某种规则装入到 Cache 中,并建立主存地址与 Cache 地址之间的对应关系。主存对 Cache 使用相联式访问,Tag SRAM 和 Cache RAM 构成相联存储器。相联存储器的每一个存储块都有额外的存储信息来标识该块的地址和状态,称为标签(Tag)。存储标签的存储器叫做 Tag RAM (见上一节)。当访问相联存储器时,将访问地址和每一个标签同时进行比较,对标签相同的存储块进行访问。根据主存和 Cache 的相联关系,Cache 可分为三种基本结构。

1) 全相联 Cache

在全相联 Cache 中,存储块与块之间、存储器地址之间或存储顺序没有直接关系。主存中的任意一块可以映像到 Cache 中的任意一块的位置上,如图 6.37 所示。如果主存有 M 个块,Cache 中有 N 个块,则二者之间的映像关系共有 $N \times M$ 种。如果用 Tag RAM 存放这种关系目录表,则目录表项数为 N ,字长为 Cache 中的地址块号长度与主存地址中的块号长度之和加一位。当访问数据时,Cache 控制器把访问地址和表中所有的主存地址块号相比较,进行确认。这种 Cache 结构命中率高,缺点是比较地址需要较长的时间,速度较慢。

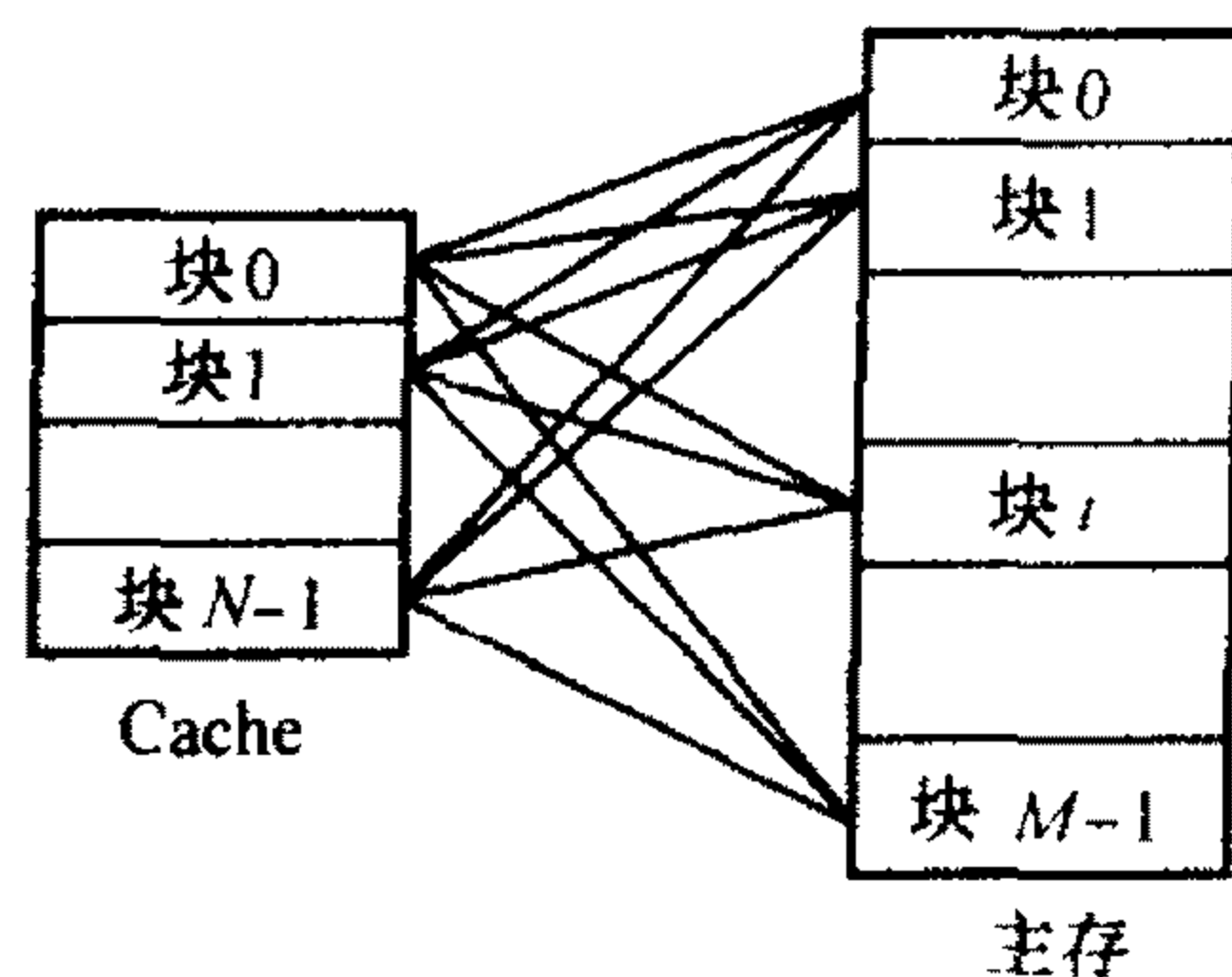


图 6.37 全相联映像方式

2) 直接映像 Cache

使用直接映像 Cache,地址只需要比较一次,规定主存储器的一块只能映像到 Cache 中一个特定的块中,每个块位置在 Cache 中分配一个索引字段,用字段区存放在 Cache 位置上的不同的块号。单路直接映像把主存储器分成若干页(区),主存储器的每一页与 Cache 存储器的大小相同,主存每页的块数与 Cache 的总块数相等。直接映像只能把各页中相对块号(又叫做偏移量)相同的那些块映像到 Cache 中同一块号的特定块中。如:主存页中的块 0 只能映像到 Cache 的块 0 中……相对于全相联 Cache,直接映像 Cache 查找速度快,但容易产生冲突,当主存中不同页的块号相同的块都映像到 Cache 同一块中,而这些块又都是当前的常用块时,就要在页之间频繁调用,命中率大大降低,且不能充分利用 Cache 的空间。

3) 相联 Cache

组相联 Cache 是全相联 Cache 和直接映像 Cache 之间的一种结构。这种结构的 Cache

使用了几组直接映像的块,主存也对应地分成若干与 Cache 分组大小相同的组,从主存组到 Cache 组之间采用直接映像方式。当组映像关系建立后,两个对应的组内部采用全相联方式。对于某一个特定的块号,可以允许有几个块位置,因而可以提高命中率和系统效率。

6.5.2 Cache 与 DRAM 的存取一致性

Cache 中应尽量存放 CPU 最近一直在使用的指令与数据。当 Cache 装满后,可将长期不用的数据删除,提高 Cache 的使用效率。为保持 Cache 中数据与主存储器中数据的一致性,同时避免 CPU 在读/写过程中遗失新数据,确保 Cache 中更新过的数据不会因覆盖而消失,必须将 Cache 中的数据更新及时并准确地反映到主存储器。这个问题的解决有以下几种方式。

1. 贯穿读出式(Look Through)

该方式的原理示意图如图 6.38 所示。



图 6.38 贯穿读出式原理示意图

在这种方式下,Cache 处在 CPU 与主存之间,CPU 对主存的所有数据请求都首先送到 Cache,由 Cache 自行在自身查找。如果命中,则切断 CPU 对主存的请求,并将数据送出;如果不命中,则将数据请求传给主存。该方法的优点是降低了 CPU 对主存的请求次数,缺点是延迟了 CPU 对主存的访问时间。

2. 旁路读出式(Look Aside)

这种方式的原理示意如图 6.39 所示。

在这种方式中,CPU 发出数据请求时,并不是单通道地穿过 Cache,而是向 Cache 和主存同时发出请求。由于 Cache 速度更快,如果命中,则 Cache 在将数据回送给 CPU 的同时,还来得及中断 CPU 对主存的请求;若不命中,则 Cache 不做任何动作,由 CPU 直接访问主存。它的优点是没有时间延迟,缺点是每次 CPU 都要访问主存,这样,就占用了部分总线时间。

3. 写穿式(Write Through)

任一从 CPU 发出的写信号送到 Cache 的同时,也写入主存,以保证主存的数据能同步地更新。它的优点是操作简单,但由于主存的速度慢,降低了系统的写速度并占用了部分总线时间。写穿式原理示意图如图 6.40 所示。

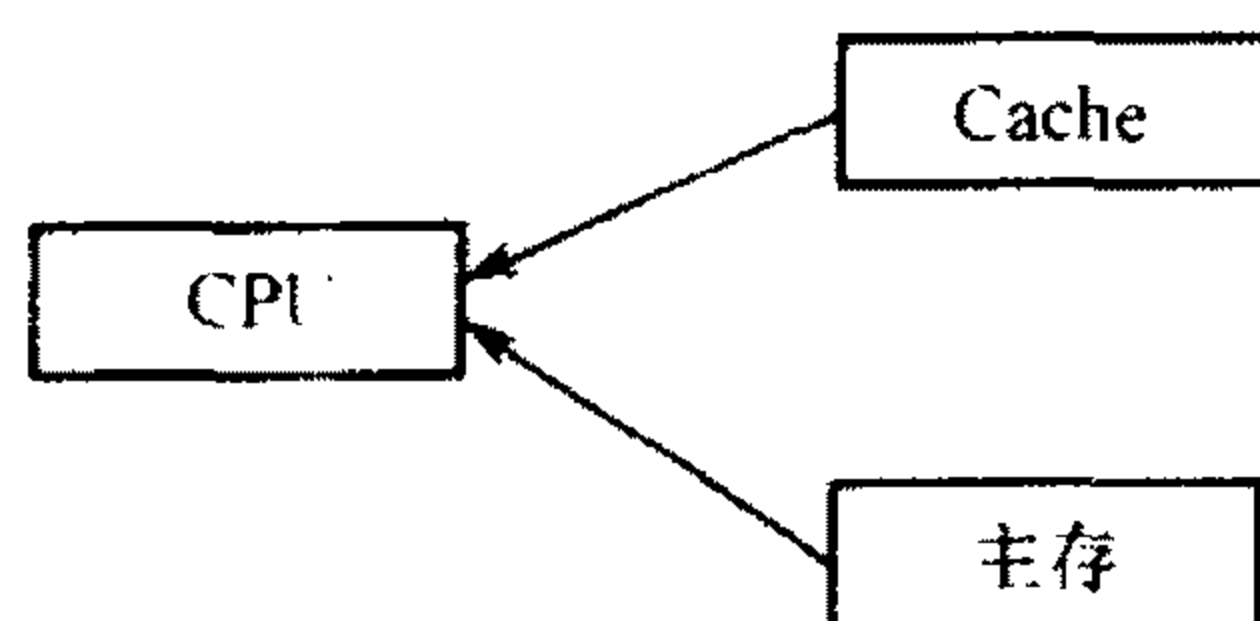


图 6.39 旁路读出式原理示意图

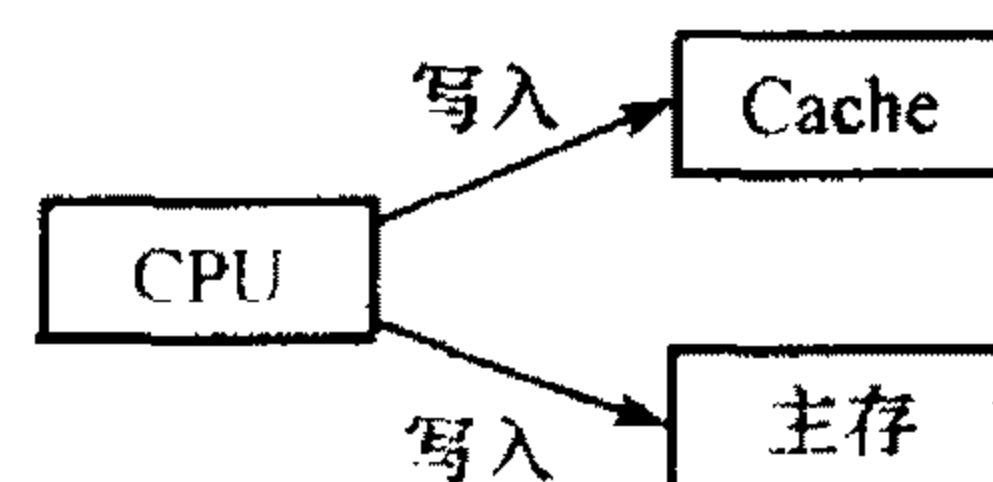


图 6.40 写穿式原理示意图

4. 回写式 (Write Back)

为了克服写穿式中每次数据写入都要访问主存、从而导致系统写速度降低并占用总线时间的弊病,尽量减少对主存的访问次数,就有了回写式。回写式原理示意如图 6.41 所示。它的工作原理是这样的:数据一般只写到 Cache,而不写入主存,从而使写入的速度加快。但这样有可能出现 Cache 中的数据得到更新而对应主存中的数据却没有变(即数据不同步)的情况。此时可在 Cache 中设一个标志地址及数据陈旧的信息,只有当 Cache 中的数据被再次更改时,才将原更新的数据写入主存相应的单元中,然后再接受再次更新的数据。这样保证了 Cache 和主存中的数据不致产生冲突。

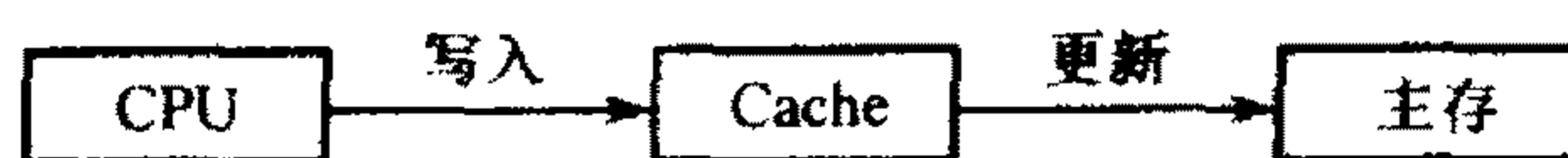


图 6.41 回写式原理示意图

6.5.3 Cache 的分级体系结构

一个微处理器的性能通常由如下几种因素估算:

$$\text{性能} = \frac{kf}{CPI - (1 - H) \times N} \quad (6.6)$$

式中: k 为比例常数, f 为工作频率, CPI 为执行每条指令需要的周期数, H 为 Cache 的命中率, N 为存取周期数。

显然,为了提高处理器的性能,应尽量提高工作频率 f ,减少执行每条指令需要的周期数 CPI ,提高 Cache 的命中率 H ,减少存取周期数 N 。要达到这些目的,可采用以下技术:

- ① 同时分发多条指令和采用乱序执行,可以减少 CPI 的值;
- ② 采用转移预测和适当增加 Cache 容量,可以提高 H 值;
- ③ 采用高速的总线接口和不分块的 Cache 方案,可以减少存取周期数 N ;
- ④ 采用指令数据预取技术,可以提高 Cache 的命中率 H 。

有时仅采用一个级别的 Cache 还不能满足要求,而需要增加第 2 级 Cache,这就构成了 Cache 的分级结构。对于一个有多级 Cache 的微型计算机系统,其一级缓存(L1 Cache)是集成在 CPU 内部的,二级缓存(L2 Cache)的设计分芯片(卡匣)内置和外置两种。一般来讲,80%的内存申请都可在一级缓存中实现,即在 CPU 内部就可完成数据的存取,另外 20%的内存申请中的 80%又可只与二级缓存打交道。因此,只有 4%的内存申请定向到主存 DRAM 中。

过去,Cache 分级结构的不足在于高速缓存组数目受限,需要占用线路板空间和一些支持逻辑电路,使成本增加。分级 Cache 结构一般分为二级,即一级 Cache (L1 Cache)和二级 Cache(L2 Cache)。

现在,新型的 CPU 除将 L1 Cache 与 CPU 集成在一起外,对 L2 Cache 也均采用了芯片(卡匣)内置设计,其工作频率与 CPU 内核的频率相同。如 Intel 的 PII/PIII、新 Ceron 及后来的产品系列。这种设计使系统的性能进一步得到提高。L1 Cache 的容量在 8 KB ~ 64 KB 之间,L2 Cache 一般比 L1 Cache 大一个数量级以上,其容量从 128 KB ~ 2 MB 不等。为了减少 Cache 的冲突,新型的 CPU 往往还把 L1 Cache 分为指令 Cache 和数据 Cache,使指令和数据的访问互不影响。

在 80386 以上档次的 CPU 中,已普遍采用了高速缓冲存储器技术。例如,在 80486DX 芯片上,有一个供指令和数据共用的 8 KB 高速缓存。在 Pentium II 系列以上的 CPU 内部都有 8 KB ~ 64 KB 的一级 Cache 和封装在 CPU 内部的二级 Cache。

随着计算机技术的发展,CPU 的主频已越来越高,系统构架越来越先进,而主存 DRAM 的结构和存取时间缩短的进程则相对较慢。因此 Cache 技术就愈显重要,结果使得在微型计算机系统 Cache 越来越大。现在已把 Cache 的容量和速度做为评价和选购微型计算机系统的一个重要指标。

6.6 存储器管理技术

6.6.1 虚拟存储器的实现机制

在高性能的计算机系统中,外存储器(硬盘)成为存储系统构架的第三个层次,充当前两个层次(CPU 内部的高速缓存和主存)的后援,故也称为“辅存”。在存储器管理硬部件和操作系统中存储管理软件的支持下,将主存和辅存的地址空间统一编址,使用户获得一个很大的编程空间,好像可以使用一个比物理内存大得多的存储器,被称为虚拟存储器(Virtual Memory)。这样的虚拟存储技术对用户来说,自然是极有价值的,特别是大容量、高密度的声

音、影像等多媒体信息的传输,没有大容量的存储技术支持是不可能实现的。

在虚拟存储的概念中,通常将程序中出现的逻辑地址称为“虚地址”,而实际主存的地址是物理地址。一般地,虚拟地址的位数远大于物理地址的位数。程序运行时,将程序由辅存装入主存供 CPU 执行,就必须进行虚地址到实地址的转换,把程序地址转换或映射为物理存储器地址。

对于操作系统的编制者,需要考虑这样一些问题:主存空间与辅存(磁盘)空间如何分区管理,如何将虚拟地址映射到物理地址空间,虚拟—实地址转换采用什么替换算法等。相应地可分为三种方式:页式、段式和段页式虚拟存储器。在高档微处理器中,已将有关的存储管理硬件集成在 CPU 芯片之内,可支持操作系统选用上述方式之一。下面简要介绍这三种管理方式。

1. 段式管理

一个大程序往往按逻辑结构(如划分为模块)分为若干段,各段大小可变。相应地,段式管理也随程序的需要动态地分段,并将各段的起始地址与段的长度写入段表之中。编程使用的虚地址就包含两部分:高位是段号,低位是段内地址。如 80386 的分段模式,使用具有两个部分的虚拟地址,即段号部分及偏移量。段号部分是指 CS、DS、SS、ES、GS、FS 个段。运行时按段进行虚实地址转换。段式管理要在内存中建立段表。每一程序段在段表中都占一个表目,如图 6.42 所示。表目的顺序是虚段号的自然顺序。虚段号不占表目空间,体现的是表目地址,用虚线框标出,图中程序有 3 个段装入内存的不同区域。段表中记录了各段在内存的首地址。表中的装入位指明某段是否已装入内存。若已装入内存,从段表中读出该

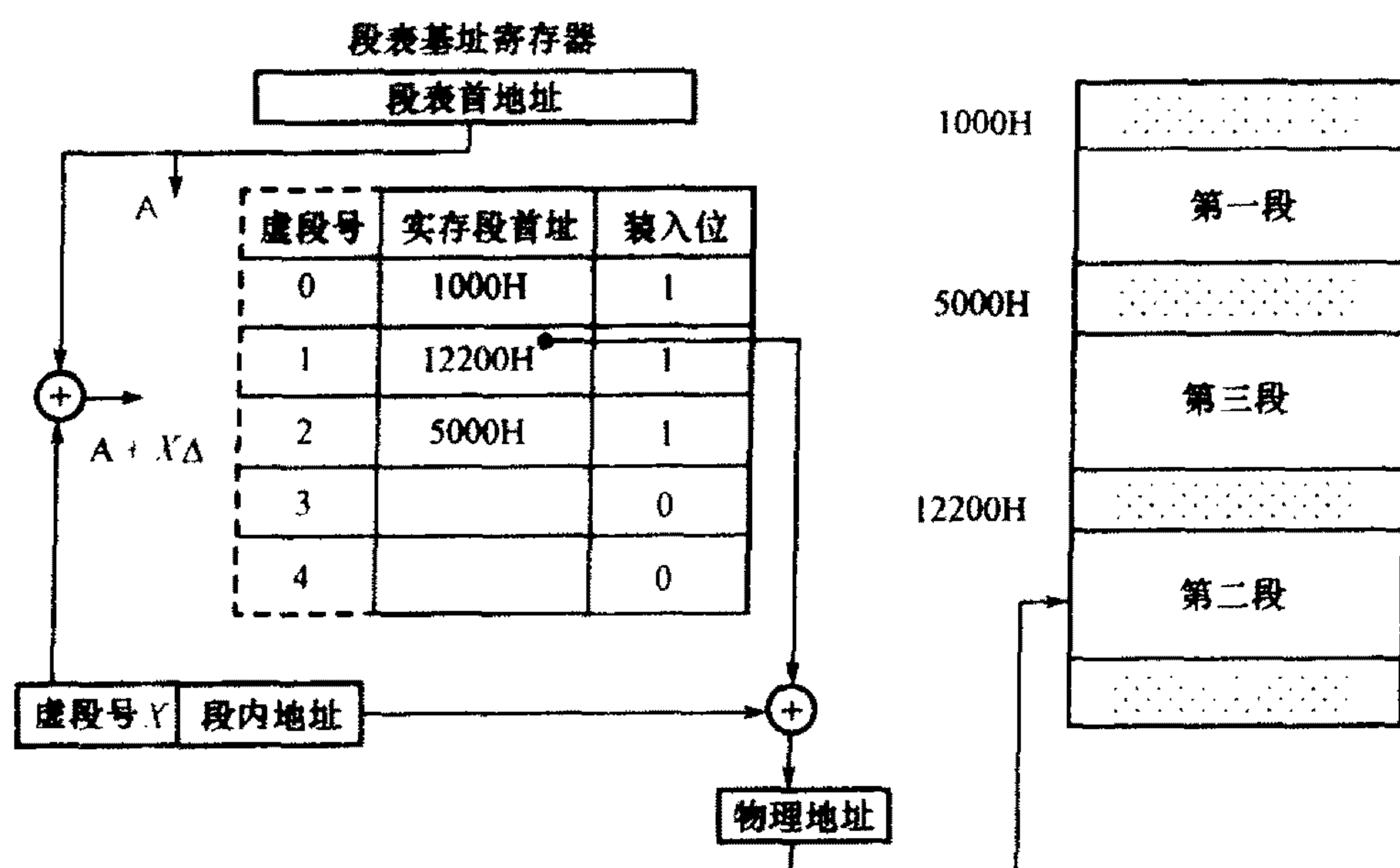


图 6.42 段式管理

段在主存中的起始地址,与段内地址(偏移量)相加,得到对应的物理地址。

80386 为地址偏移部分提供了灵活的机制,使用存储器操作数的每条指令规定了计算偏移量的方法,这种规定叫做指令的寻址方法,8 个通用寄存器的任一个都可用作基址寄存器(基地址也可以不要),除堆栈指针外的七个寄存器又可用做变址寄存器,再把这个变址寄存器的值乘以 1、2、4、8 中的任一个因子,然后再加上一个 32 位的偏移量作为地址的偏移部分($32\text{ 位的偏移量} = \text{基地址} + [\text{变址寄存器的内容}] \times \text{比例因子} + \text{位移量}$,例如 $\text{DS: EDX} + [\text{ESI}] \times 8 + \text{位移量}$)。这种寻址方式提供强有力而又灵活的寻址机制,非常适用于高级语言。

段是形成虚拟—实地址转换机制的基础。每个段由 3 个参数定义:

① 段的基地址 即线性空间中段的开始地址。基地址是线性地址空间对应于段内偏移量为 0 的虚拟地址。

② 段的界限(Limit) 指段内可以使用的最大偏移量,它指明该段的范围大小。

③ 段属性 如可读或写入段的特权级等。

以上 3 个参数都存储在段的描述符中,而描述符又存于段描述符表中,即描述符表是描述符的一个数组,而虚拟—实地址转换时要访问描述符。

2. 页式管理

页式管理面向存储器物理结构。其特点是所管理的主存和辅存的存储空间分为固定的大小,称为“页”。主存的页为实页,辅存的页为虚页。页的大小一般为 4 KB ~ 512 KB,用户编程时,将程序的逻辑空间分为若干虚页。程序所用的虚地址分为虚页号和页内地址。由于页面大小相同,虚、实地址中的页内地址部分相同。

在主存中建立一种页表,提供虚实地址变换依据,若计算机采用多用户多道程序工作方式,页式管理在内存中为每个用户设置一个页表,页表按虚页号顺序排列,来登记虚地址各页在实地址的位置。表 6-8 给出了一种页表组织示例,表中每行记录了与某个虚页对应的有关信息。虚页号由程序的虚地址给出,盘号(块号)是该页在磁盘中的起始位置,控制位通常包含:装入位记录该页是否被装入主存,为 1 表示已装入;修改位,指明对应的实存页是否被修改过;读/写保护位,指明该页的读/写允许权限等。实页号,如果该虚页在主存中,则登记对应的主存储器的页号。

表 6-8 页表示例

虚页号	盘号(块号)	控制位	实页号
0	10	1	34
1	25	0	⋮
⋮	⋮	⋮	⋮

由于每一页的长度是固定的,因此,不需要像段式虚拟存储器中的段界限这个参数字段,只需实页号这一个字段即可标明主存地址。

在 CPU 内部有个基址寄存器堆,用来存放每张页表的基地址,每个用户(每道程序)使用其中的一个基址寄存器。当 CPU 根据虚地址访存时,通过多用户虚地址中的用户号可以直接找到与用户程序相对应的基址寄存器,进而得到相应的页表的起始地址。根据页表上该行内容判断该虚页是否在主存中。若已调入内存,可从页表中读出相应的主存实页号,在将其与页内地址合成,得到对应的主存实地址,据此可以访问主存物理单元。

若装入位为“0”,即该虚页未调入,则产生缺页中断,执行替换算法,将缺页调入。

3. 段页式管理

上述的分段和分页虚拟存储管理机制,它们都是使用建立在存储器中的各种表格,规定各自的转换算法。这些表格只允许操作系统进行访问,而应用程序不能对其修改。操作系统为每一个任务维护一套各自不同的转换表格,其结果是每一任务有不同的虚拟地址空间,并使各任务彼此隔离开来,以便完成多任务分时操作。另外,二者在使用上还各有利弊。

页式管理是面向存储器本身物理结构来分页,有利于存储空间的利用与调度,但页的大小固定,而程序的实际长度一般是不固定的,一页通常不能表示一个完整的程序段功能。即页可能与程序中的逻辑段交叉,不便于程序的执行、保护和共享。

段式管理面向程序的逻辑结构来分段,段的大小、位置可变,以段为单位进行各种处理,利于程序的编译、执行、保护和共享。但段的大小可能差异很大,使得连续装入的各段首、尾地址没有规律。新、旧段更迭时会形成存储垃圾区,降低存储空间的利用率。

为综合两种方式的优点,许多计算机采用段页式管理方式。以 80386 为例,80386 先使用分段机制,把包含两个部分的虚拟地址空间转换为一个中间地址空间的地址,这一中间地址空间称为线性地址空间,其地址称为线性地址。然后再用分页机制把线性地址转换为物理地址,如图 6.43 所示。

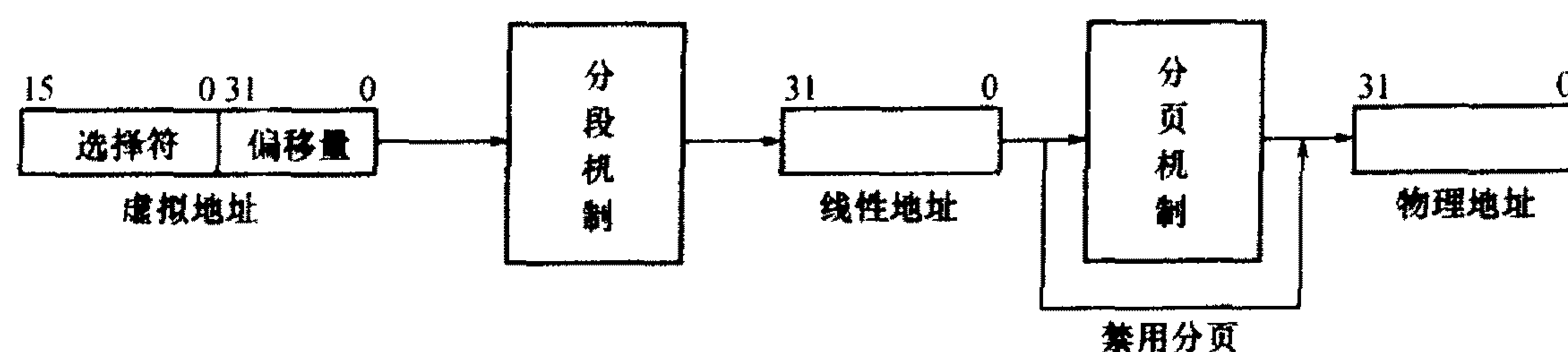


图 6.43 虚拟 - 物理地址转换

虚拟地址空间是二维的,它所包含的段数最大可达 16 K 个,每个段最大可达 4 GB,从而

构成 64 TB 容量的庞大虚拟地址空间。线性地址空间和物理地址空间都是一维的,其容量为 $2^{32} = 4 \text{ GB}$ 。事实上,分页机制被禁止使用时,线性地址就是物理地址。

段页式虚拟存储器兼有页式与段式的优点,但要经过两级查表才能完成地址转换,费时较多。

虚拟存储系统是在存储体系层次结构(辅存—内存—高速缓存)基础上,通过存储器管理部件 MMU,进行虚拟地址和实地址自动变换而实现的。有了这样的大容量内存的管理手段,就可支持多用户多任务的操作系统。从微处理器管理内存的角度来说,虚拟存储器管理机制也叫做虚地址保护方式。

所谓保护有两个含义,一是每一个任务分配不同的虚地址空间,使任务之间完全隔离,实现任务间的保护。二是任务内的保护机制,保护操作系统存储段及其专用处理寄存器不被用户应用程序所破坏。

通常操作系统存储在一个单独的任务段中,并被所有其他任务共享,每个任务有自己的段表和页表。

在同一任务内,定义 0~3 四种特权级别,0 级最高。定义为最高级中的数据只能由任务中最受信任的部分进行访问。特权级可以看成 4 个同心圆,内层最高,外层最低,特权级的典型用法是把操作系统的核心放在 0 级,操作系统的其余部分放在 1 级,而应用程序放在 3 级,留下的部分供中间层软件用,如图 6.44 所示。不同的软件只能在自己的特权级中运行,这样就可避免应用程序对操作系统内核的破坏,使系统运行更加可靠。

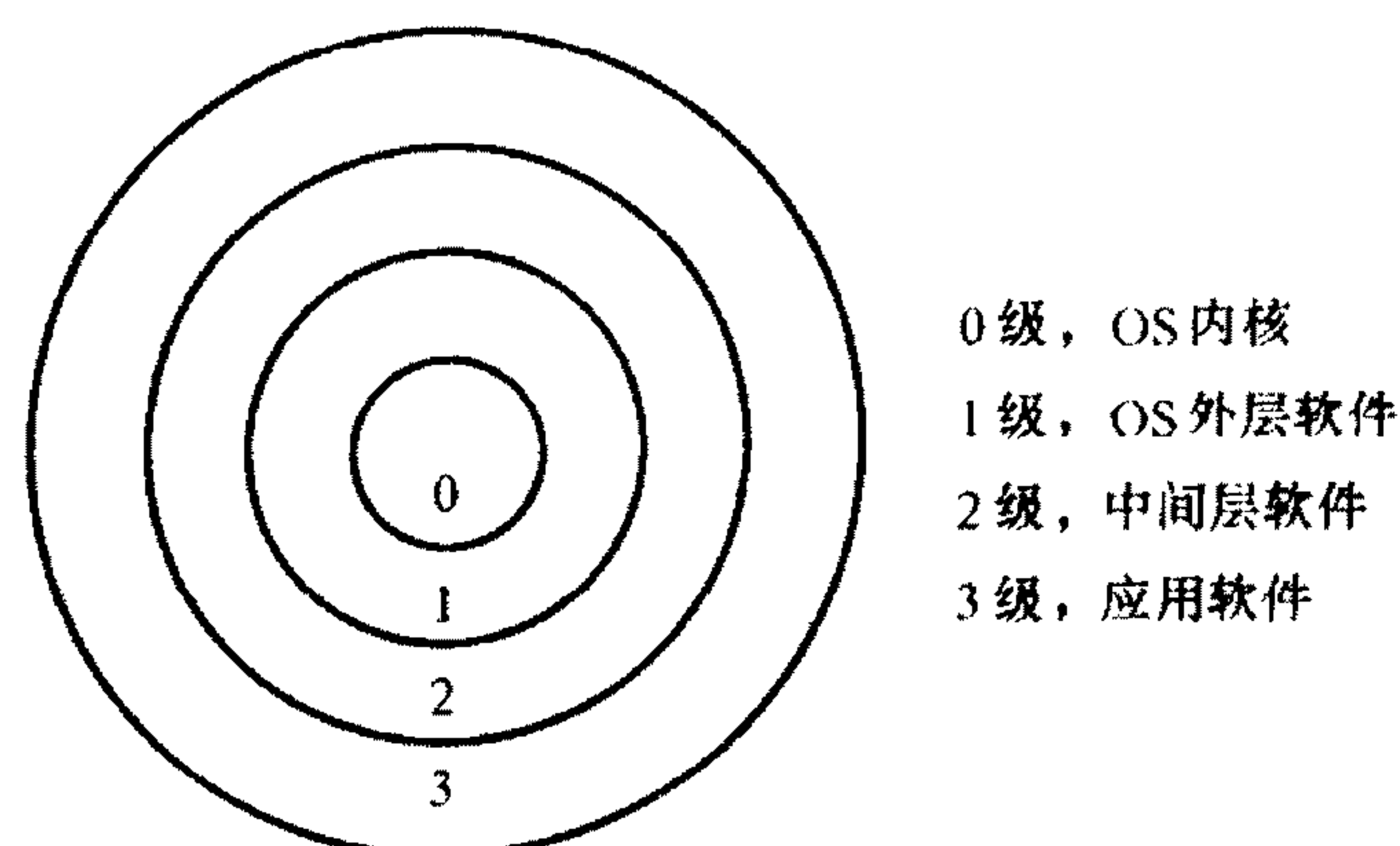


图 6.44 80386 的特权级(Ring0 ~ Ring3)

此外,80386、80486 和 Pentium 系列微处理器还支持虚拟 8086 方式和实地址方式:

① 虚拟 8086 方式 这是 80386、80486 和 Pentium 的一种新的工作方式,该方式支持存储管理,保护及事务环境中执行 8086 程序。它创建了一个在虚拟 8086 方式下执行 8086 程序的任务的环境,使操作系统能够运行 DOS 下的程序。

② 实地址方式 这是 80286 到 Pentium 微处理器最基本的工作方式,即:这些系列的处理器在启动时都采用实地址方式,与 8088/8086 工作方式基本相同,寻址范围只能在 1 MB 范围内。复位时,启动地址为 FFFF0H,此地址通常安排一个跳转指令,转至上电自检和自举程序。另外,地址为 0~003FFH 的内存区域保留为中断向量区。可以认为实地址方式只使用低 20 位地址线,寻址 1 MB 内存空间,与 8088/8086 工作情况是一致的。启动之后,根据所安装的操作系统,可以改变这种操作方式。80386(或以上)的指令系统中除了 9 条保护方

式指令外,其余均可在实地址方式下运行。在实地址方式下,8086 系统对存储器是按分段的方法进行管理的(详见前面的有关章节)。

6.6.2 Windows 9X 的内存管理

Windows 系统采用虚拟内存管理机制。在 32 位机系统中,Windows 95 给应用程序的设计者提供了 32 位平面式存储器管理方式。理论上,Windows 允许每一个 Win 32 进程可以访问 4 GB 的存储器地址空间,显然,这是虚地址空间。(同时,这样的机制选择 4 GB 的空间为一个大致,相当于分段被禁止)从 0~2 GB 的地址空间分配给各个 Win 32 进程,应用程序的所有存储器请求都被映射到这一区域。应用程序设计者编程时只考虑虚拟存储器单元,由 Windows 操作系统将虚拟地址映射成物理单元(RAM 或是分页文件)。

Windows 的存储器管理规则将 4 KB 作为一页,当自由 RAM 不够用时,存储器管理程序就将应用程序或数据文件中的某些页复制到分页文件(又称为交换文件)中,从而释放某些系统存储器。交换文件实际上是建立在硬盘上的一个隐含文件,分为临时交换文件和永久交换文件。临时交换文件为 WIN386.SWP,它随 Windows 系统的启动而产生,随着退出 Windows 系统而消失,在 Windows 9X 系统中,它位于 C:\Windows 目录下;永久交换文件为 386SPART.PAR,它不管 Windows 运行与否都存在于硬盘上。一般情况下,使用更多的是临时交换文件,但在硬盘空间允许的情况下,使用永久交换文件更好一些,因为它在磁盘上以连续方式存放,存取速度比临时文件快得多。

在页交换文件中,存储器是以 4 KB 为单位构造的,分页文件采用 4 KB 为访问对象。当一个 Win 32 进程要定位存储器页时,它需要通过标准的 API 接口对操作系统提出请求,操作系统负责确定这些页的位置是在主存储器中还是在分页文件中。当定位一页时,存储器管理程序就在进程页表中增添一个新项。如果该新页在 RAM 中(存放位置为 1),其余的 31 位就包含该页的所有说明信息;反之,如果查找的页不在 RAM 中,存在位就等于 0,而其余 31 位都是“空”。

访问这个空项将引起一次“异常”,接着就要将该页重新装入一个新的存储器位置中,操作系统将该新页的地址复制到页表项中,同时相应地修改存在位为“1”。再重复执行前面引起异常的指令。在 Windows 95 中,32 位页虚地址实际上分为 3 部分:10 位、10 位和 12 位。见表 6-9。

表 6-9 分页方案

31	...	22	21	...	12	11	...	0
页目录项			页表项			偏移量		

当操作系统生成一个进程时,主存给它分配一块存储器,并填入包含页目录地址在内的有关信息。从左开始的前 10 位是页目录项,一个页目录包括 1 024 项,紧接着是页表项,最后是偏移地址。而偏移地址是“真实”的,即标示偏移地址的低 12 位将与根据分页算法得到的页帧地址相加,而指向一页中的任何一个字节。

当进程激活时将页目录地址装入三号控制器(80386 以上的处理器的控制器,其中 CR3 存放页目录基地址)根据页目录值找到页表目录(32 位)。该目录的前 20 位称为页帧地址,通过这 20 位地址,可访问页表中的一项,请记住,4 GB 的空间包含了一百万页($1\text{ M} \times 4\text{ 096 B} = 4\text{ GB}$)。选择了页表,系统就要处理表 6-9 中的第 2 部分,即页表项,页表的结构和页目录相同(见表 6-10),通过读页表项的前 20 位,分页算法最终指定包含进程实际数据或代码的页(称为页帧)。也就是虚地址变换成实地址。表 6-9 中 32 位偏移量的最后一部分是 12 位页内偏移值本身,它使得系统能找到一页中的任何一个字节。这个方案的寻址过程如图 6.45 所示。

表 6-10 一个页表项

标志	占位	说 明
P	1	Present (存在)
R/W	1	Read/Write (读/写)
U/S	1	User/Supervisor (用户/管理员)
PWT	1	Page Write Transparent (透明写页)
PCD	1	Page Cache Disable (禁止页高速缓存)
A	1	Accessed (被访问的)
D	1	Dirty (页面重写标志)
AVAIL	3	Available (可用的)

整个寻址方案至少涉及到图 6.45 中的 3 页(页目录、页表和页帧)。

每当计算一个地址时,并不能肯定每一页都存在于物理存储器中,特别是对于只有少量 RAM(4 MB)的 Windows 95 系统,很多页都驻留在分页文件中(存在位置为 0),对于一个简单的操作系统,可能出现 3 个页异常;当发生页异常时,需要从磁盘存储器中调入一页到主存储器中,而多用户的虚页数比主存储器的实页数多得多。因此,必然会出现主存中所有页面都被占用,而又要从磁盘存储器中调入新页的情况。这时,必须从主存储器中淘汰一个不常用的页面(或程序段),以便腾出主存空间,操作系统使用页面替换算法解决这一调度问题。

Windows 突破了 DOS 的 640 KB 的常规内存的局限,在内存的管理上有了很大的进步。但 Windows 9X 在内存管理上仍有很多不足的地方:第一,它允许程序占用前 1 MB 的内存,而该区域实际上是为一些特殊的需求而留用的资源,如果此区域被占用,当这些特殊需求出

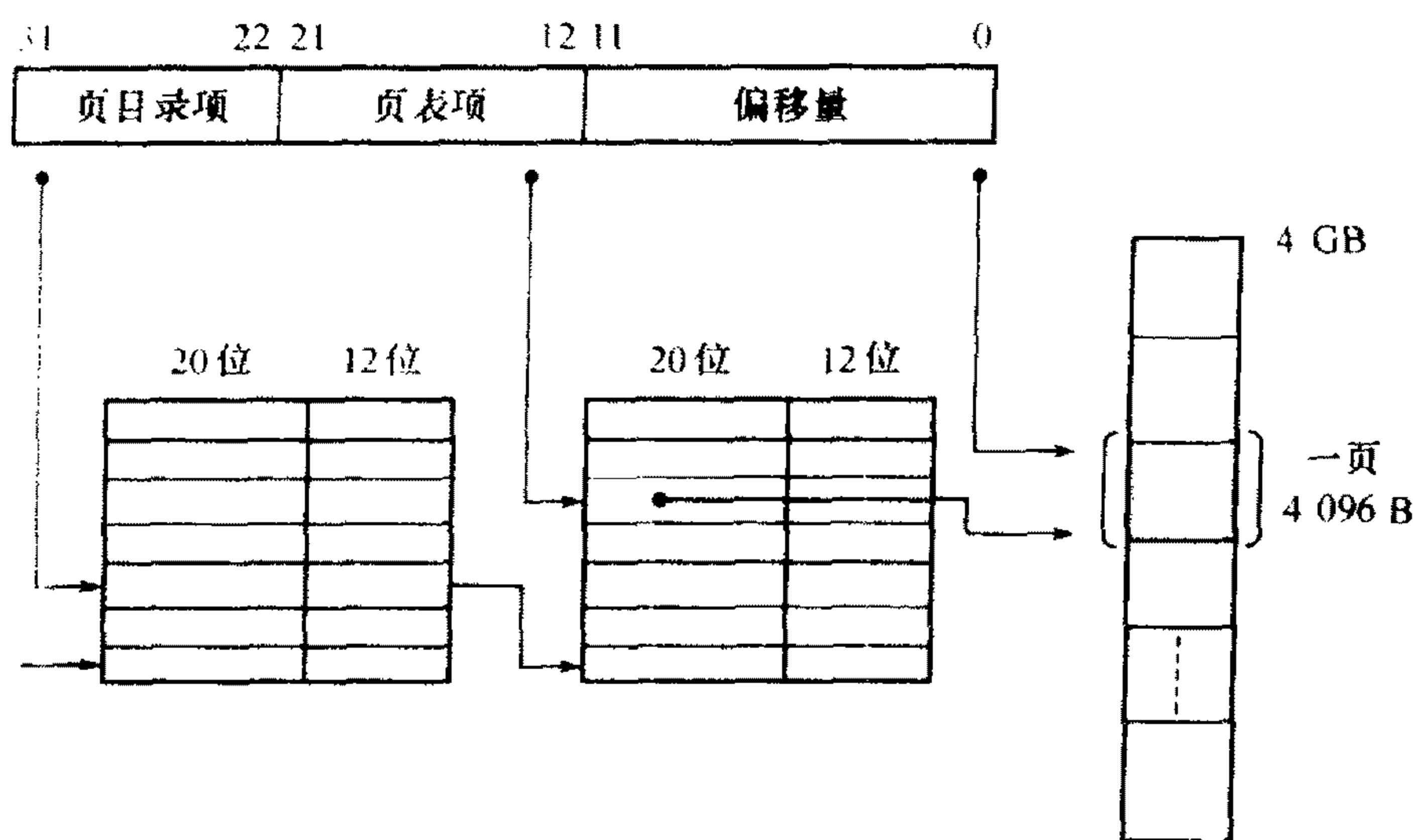


图 6.45 Windows 95 和 Windows NT 的寻址方案

现时, Windows 往往会发现资源已被耗尽; 第二, Windows 需保持跟踪大量的、自身需要的以及正在运行当中的 Windows 应用程序的用户界面信息, 这样, 在系统资源局部堆栈中就难有足够的空间去容纳有可能想要运行的所有应用程序的相关信息; 第三, Windows 保持 7 层系统资源局部堆栈, 每层堆栈不超过 64 KB。每个堆栈都包含 Windows 系统运作的重要信息, 当其中任何一个堆栈的自由空间少于 30% 时, 系统的响应速度就会明显降低, 并出现“Out of Memory”的错误, 实际上此时可能还有数以兆计的系统内存没有被使用。如果某个应用程序或驱动程序不正确地使用了资源栈, 而导致堆栈被充满时, Windows 就会崩溃, 即出现一般保护性错误 GPF (General Protection Failure), 此时只能重新启动机器了。如某个软件或硬件使用了其他软件或硬件占用的内存时, 就会导致 GPF 发生。当系统中增加了一个新的应用程序、一个带有驱动程序的硬件或是重新对系统进行了配置时, 最易产生 GPF 错误。其中, 显示驱动程序往往是 GPF 的主要根源之一。

目前, 有一个流行的软件——“优化大师”, 其中附有整理内存碎片, 优化内存分配的功能。读者在遇到系统响应速度不尽人意时, 可试着从网上下载该软件, 一般会有收效。

6.7 新一代内存条的硬件技术发展

6.7.1 DRAM 的发展

CPU 芯片的飞速发展使得 DRAM 必须不断地改进才能适应其发展的需要。CPU 芯片

由 8 位、16 位发展到 32 位。目前,新型的 64 位微处理器也即将推向市场。时钟频率由最初的不到 5 MHz 发展到今天的 2 GHz 以上。随着所需内存的不断扩大,传统的 DIP 内存芯片已被内存条所取代,与 CPU 时钟相适应的内存速度,也从最初的要求为 200 ns 直到今天的 5 ns 甚至以下才能满足需要。这些要求对 SRAM 较易达到。但在微型计算机应用中,与用户关系最密切的是 DRAM,近年来,随着微型计算机速度的不断提升,内存条的种类也经历了从 PM DRAM、EDO DRAM 到目前流行的 DDR SDRAM、RDRAM 的发展历程。下面对这些品种的 DRAM 进行简单的介绍,使读者对流行的内存条有所了解。

1) FPM RAM

早期的 PM RAM 称为“页面式 DRAM”(Page Mode DRAM, PM DRAM)。因为通常把 DRAM 存储地址空间按页面划分进行访问。最初的 PM RAM 存取时间较慢,一般在 120 ns 左右。为了提高速度,PM RAM 有了很大改进,并将改进后的 PM RAM 称为快速页面 DRAM (Fast Page Mode DRAM, FPM DRAM)。FPM DRAM 的存取时间为 80 ~ 100 ns,它一度曾是 486 微型计算机中的主流配置,但由于其速度较低,在后来的 Pentium 系列微型计算机中已很少使用。

2) EDO DRAM

EDO DRAM 是扩展数据输出存储器(Extended Data Output DRAM)的简称。从前面的知识了解到,程序运行时,CPU 可能连续访问相邻的内存单元。EDO DRAM 的特点就是在每次访问后,在把数据发送给 CPU 的同时去访问下一个页面,从而取消了页面切换所需的额外时钟周期,加速了对邻近地址单元的访问。EDO DRAM 是早期 Pentium 微型计算机的主流配置。

3) SDRAM

SDRAM 是同步式 DRAM(Synchronous DRAM)的简称。SDRAM 同以前的 DRAM 有很大的区别,它在一个 CPU 时钟周期内即可完成数据的访问和刷新,即能够与 CPU 的时钟同步工作,极大地提高了存储器的存取速度。SDRAM 采用双存储体结构,当一个存储体被 CPU 访问时,另一个存储体就做好了存取的准备,两个存储体自动切换。SDRAM 工作频率目前最大可达 133 MHz,存取时间约为 6 ~ 10 ns,是 Pentium II/Pentium III 微型计算机中流行的标准内存类型配置。SDRAM 外形如图 6.46 所示。图中内存条下方的金属引脚接触片又被称为“金手指”。

4) DDR SDRAM

DDR SDRAM(Double Data Rate Synchronous DRAM, 双倍数据速率同步内存),简称 DDR,目前普遍应用于 Pentium 4 微型计算机。所谓双倍数据速率是指它在时钟的上升沿和下降沿都可以进行数据读/写,而使得实际带宽增加两倍。故又叫做双时钟触发读/写内存,其效率比普通 SDRAM 高一倍。DDR 的外形如图 6.47 所示。从外形看,SDRAM 和当前流行的

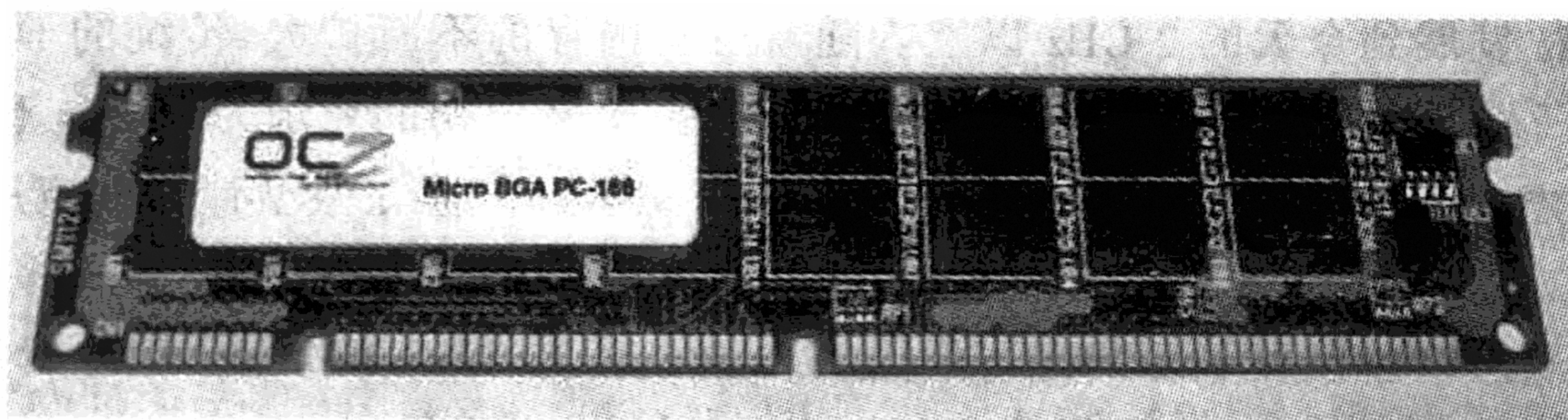


图 6.46 168 线 SDRAM 外形

DDR 内存条很接近,二者明显的不同之处是金手指的引脚,DDR 内存为 184 线;SDRAM 是 168 线。另外,SDRAM 内存条金手指上有两个缺口位置,而 DDR 内存条有一个缺口位置。

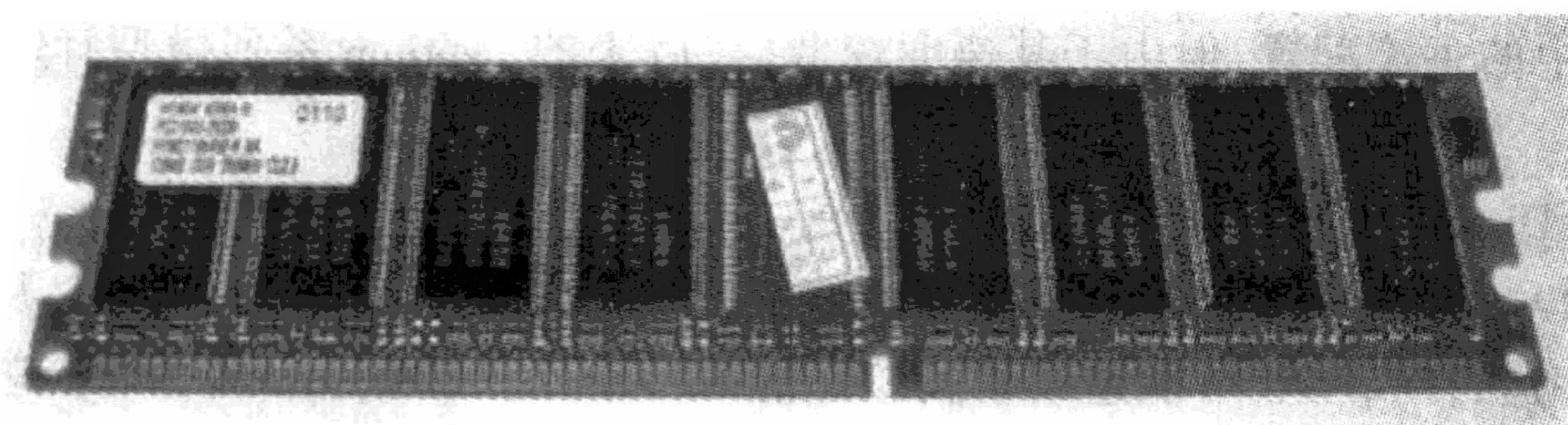


图 6.47 184 线 DDR SDRAM 外形

5) RDRAM

RDRAM 是 Rambus DRAM,即 Rambus 内存,这是 Rambus 公司开发的从芯片到芯片接口设计的新型串行结构的 DRAM,它能在很高的时钟频率范围下通过一个简单的总线传送数据。Intel 公司推广的是 Direct RDRAM,具有 300 ~ 400 MHz(现在已经扩展到 533 MHz)的工作频率,它引入了处理器设计中的 RISC(精简指令结构)思想,依靠高时钟频率来简化每个时钟周期的数据量,在时钟的上升沿和下降沿同时进行数据传送。规格有 PC600/PC800/PC1066 等,可以发现,后面的数字正好是工作频率的一倍。

RDRAM 金手指也是 184 线,但它与 SDRAM 或 DDR 内存的最大区别不在于引脚,而是 RAMBUS 罩有金属屏蔽外壳,如图 6.48 所示。这是因为 RAMBUS 工作频率高,需要通过散热片及时把芯片热量散发出去,同时金属屏蔽外壳可以有效地防止高频信号干扰。

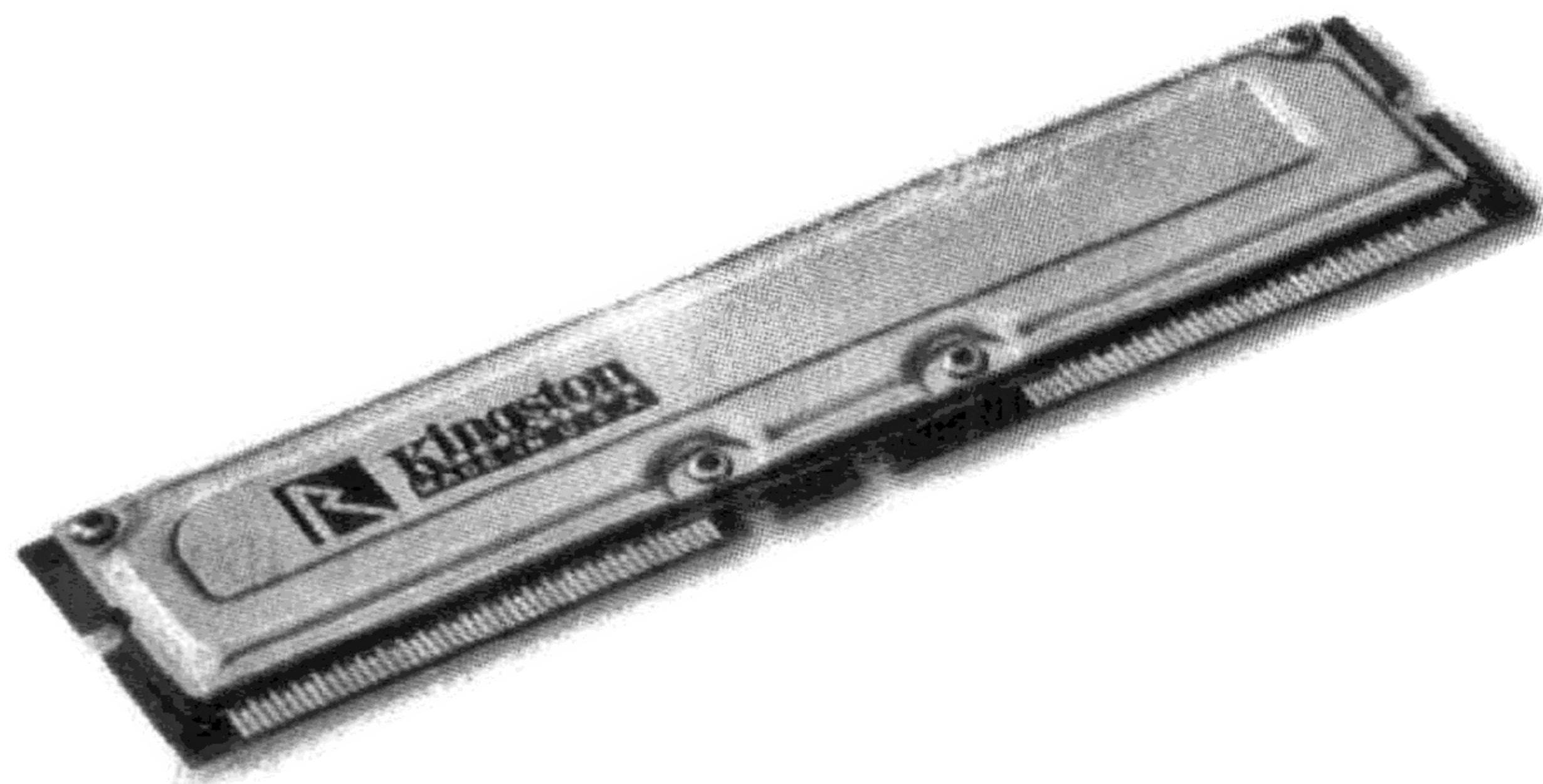


图 6.48 RDRAM 的外形

6.7.2 几种内存条的封装标准

主机板上有安装内存的插槽。常见的成品内存模块有几种封装形式,对应主机板上也有相应的内存插座形式:

1) DIP 双列直插式内存芯片,主机板对应为 DIP 芯片插座。除了早期的 PC/XT 8088 时代使用的小容量存储芯片外,早期的 80286、80386 主板常用的 DIP 芯片单片容量为 256 KB、512 KB、1 MB、2 MB 等几种。由于 DIP 内存芯片在主机板上占用面积大而存储容量小,现在通用微型计算机上已不再采用。

2) SIMM(Single In-line Memory Module,单边接触直插式存储器模块),又称 SIMM 内存。SIMM 内存条是一条焊有多片 DRAM 芯片的印刷电路板,采用统一的尺寸规格和引出线标准,常见的有 30 线和 72 线两种结构。30 线即是该内存条有 30 根引脚。30 线 SIMM 内存条常见的容量有 256 KB、512 KB、1 MB、2 MB 和 4 MB;72 线 SIMM 常见容量由 4 MB、8 MB、16 MB 和 32 MB 等。30 线 SIMM 内存条主要应用于 80486 以前的微型计算机,72 线 SIMM 内存条主要应用于 80486 和早期的 Pentium 微型计算机。

3) DIMM(Dual In-line Memory Module,双边接触直插式存储器模块),又称 DIMM 内存条。DIMM 内存条的结构与 SIMM 内存条相像,只是这种内存条的印刷电路板两面都有引脚接触片,且引脚较多,而 SIMM 只在一面有引脚接触片。DIMM 条只有 168 线一种,(参见 6.7.1 中图 6.45)。故有时人们也把 DIMM 内存条称为 168 线内存,把 72 线的 SIMM 内存条称为 72 线内存。由于内存条上的芯片集成度日趋增高而体积逐渐微型化,故又称做“内存颗粒”。

4) RIMM 是最新结构的内存条,据厂家的资料,RIMM 插槽的内存条有三种方式:

① 采用 Direct Rambus DRAM 芯片的 184 脚 RIMM 内存条;

② 采用 SDRAM 芯片并包含一片 MTH 芯片的 184 脚 SRIMM 内存条;

③ 采用 SDRAM 芯片的 168 脚 DIMM(即 PC100/PC133)内存条,加载在带有 MTH 芯片的 DIMM 升级转换卡上,再插入板上的 RIMM 插槽。

6.7.3 内存条的规范

随着微型计算机系统性能的不断提高,PC 主板的频率由 100 MHz、133 MHz 发展到 166 MHz。工作频率越高,对内存等产品设计和制造的要求也越严格,当工作频率低于 30 MHz 时(在无线电频率中,定义为高频 HF),设计制造很容易满足工作要求。如果工作频率达到或超过 100 MHz(在无线电频率中,称为甚高频 VHF),芯片引脚及连接电子元件的导线,都可能因电容、电感、输入/输出阻抗而成为发射电磁波的天线。这些在设计时就需要加以考虑平衡,以求设计出合格的系统。生产内存有多个厂家,产品的兼容性也是需要考虑的。这就要求有一个设计规范供各生产厂家遵循,这个规范要定义严格的电气指标,以保证满足这个规范的产品,能够可靠、兼容地工作。2000 年之前,Intel 公司推出了 PC100 兼容性规范,其中包括主板的设计制造的规范化文件,对布线的长度、线宽、距离、间隔、印刷电路板(PCB)的层数都做了严格规定。以下简要介绍 PC100 规范的内存技术性能。

1. CAS(列地址选通)等待时间

CAS 等待时间是当一个读命令在时钟上升沿发出时至数据出现在输出端的时延,这个值一般是 2 或 3 个时钟周期,它对内存条系统的工作速度有很大的影响,显然,在同等工作频率下,CAS 等待时间为 2 个时钟周期的芯片速度更快,性能更好。

2. 额定可用频率 GUF

额定可用频率要比厂家给定的最高频率低一些,使最高可用频率与额定可用频率之间保持一定的余量,最大限度保证系统稳定地工作。反过来说,如果要求系统总线在 125 MHz、133 MHz 的频率下工作,相应地应选用 143 MHz(时钟周期为 7 ns)的内存才能保证系统稳定工作。

3. ECC 校验

在 SDRAM 内存条中,单面有 8 或 9 片 SDRAM 芯片,9 片的一片便是用于 ECC(Error Correction Code)错误纠正代码。当内存数据发生错误时,错误纠正代码就可以起到检查纠正 1 位错误的作用。

ECC 校验是 PC100 所要求的,但目前市面上很少见到带 ECC 校验的内存条,因为增加用于 ECC 的 SDRAM 芯片无疑将提高生产成本。在服务器和工作站这种要求高可靠性的系统中,应当使用带 ECC 校验的内存条。

4. SPD

SPD 即 Serial Presence Detect, 是 SDRAM 的新规范, 它是一个 8 条引脚的 EEPROM 芯片(颗粒), 通常焊接在内存条靠右边的 SDRAM 颗粒旁边, 见图 6.49。制造厂商将该内存条的所使用的颗粒的基本信息预先写入这颗 EEPROM, 使 PC 系统可以通过 SPD 读出该内存条的基本信息, 正确地识别, 从而确定对其驱动的方法。如果没有 SPD, 系统 BIOS 只能用猜测的方法来识别该内存条的基本信息, 这样做的结果往往因驱动方法不正确而造成系统不稳定。

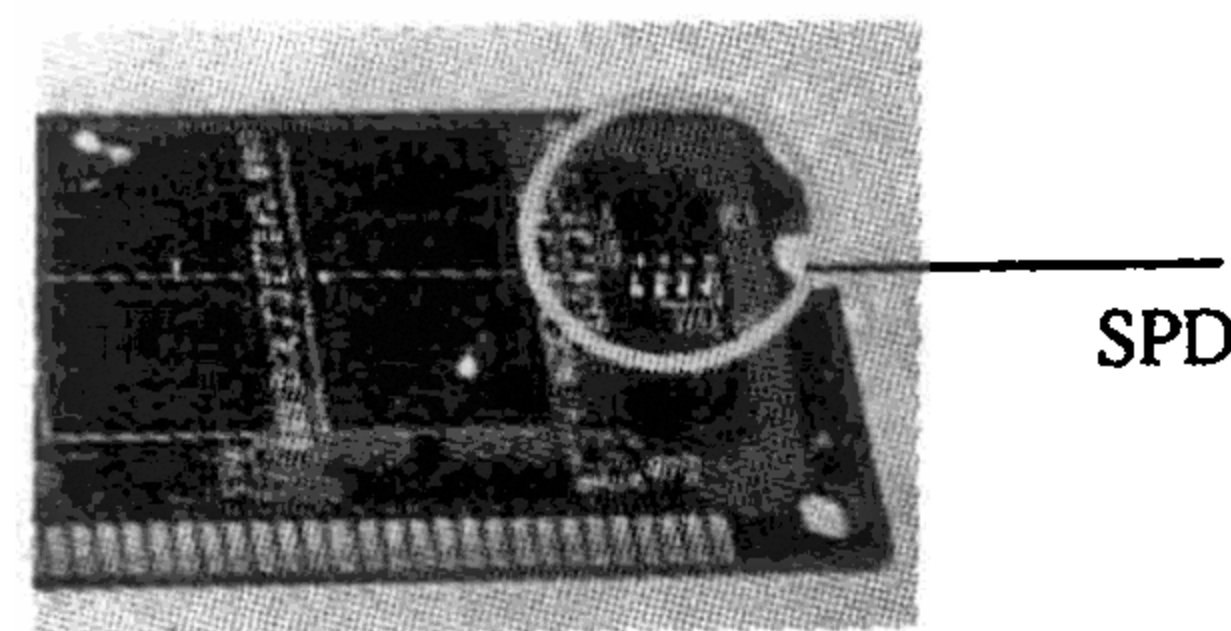


图 6.49 SPD 位置

5. 内存条 PCB 板的规定

内存条的电路板可采用 4 层或 6 层玻璃环氧基板, 但 PC100 SDRAM 内存条必须采用 6 层双面装配 PCB 设计, 6 层板设计结构依次为: 信号层、电源层、信号层、信号层、基层、基层。层间间距为 0.007 ~ 0.010 英寸。并且规定 PCB 板上的连接 Pin 必须是非斜面的镀金 Pin, 也就是“金手指”。

6. PC100 的标识规范

PC100 定义了严格的电气特性, 但没有定义标识 PC100 内存的方法, 各厂商沿用自己独立的产品代号, 没有统一的性能指标注解。可以作为识别依据的是, 满足 PC100 兼容规范的内存, 须注明 Intel 定义的产品号(P/N), 用户在选用内存时, 应认清这个 PC100 内存标记。

一般 PC100 内存的产品号表示为:

$$PCx - abc - def$$

例如, 一个典型的 PC100 SDRAM, 内存条上标注如下:

$$PC100 - 322 - 620$$

含义为:

x : 工作频率 100 MHz;

a : 最小 CAS 等待时间 = 3clk;

b : 最小 tRCD(行选通 RAS 相对于列选通的延时), 用时钟数表示, 本例为 2;

c : 最小 tRP (RAS 预充电时间), 用时钟数表示, 本例为 2;

d : 最大 tAC (相对时钟下降沿的数据读取时间), 一般是 6 ns 或 6.5 ns, 本例为 6;

e : SPD 版本号, 所有 PC100 SDRAM, 内存条上有 EEPROM, 记录内存的基本信息, 其内容记录是标准化的, 记录版本在不断更新。此处应注明内存条符合 Intel 公司的 PC100 Version 1.2;

f : 最后一位是保留值, 其值为 0。

6.8 外存储器简介

所谓外存储器就是指在微型计算机之外、通过设备接口连接的存储器。常用的外存储器包括硬盘驱动器、软盘驱动器、磁带驱动器、CD-ROM 及存储卡等。与内存相比,外存储器的容量非常大,但速度慢。内存中的信息一般在断电后,所存储的信息便立即消失。虽然 ROM 停电后能保存信息,但只能读出,不能在线写入,只能存放一些相对不变化的信息和数据,如 BIOS 等。而外存储器为计算机提供了大容量、永久性的存储功能。

6.8.1 硬盘及硬盘驱动器

硬盘的存储容量大,存取速度相对软盘也较高,是目前微型计算机系统配置中必不可少的外存储器。

1. 硬盘的基本结构

微型计算机系统中配置的硬盘均为固定盘片结构,由封装在铸铝腔体中的头盘组件(Head Disk Assembly)与控制电路印刷电路板组件(PCBA, Print Circuit Board Assembly)组成。这种结构的硬盘又称为温彻斯特磁盘,简称温盘。

2. 硬盘的工作原理

硬盘是在非磁性的合金材料或玻璃基片表面涂上一层很薄的磁性材料,通过磁层的磁化方向来存储“1”、“0”信息。盘片目前大都采用具有高密度、高剩磁、高矫顽力的金属薄膜工艺制造。

硬盘驱动器由磁盘和磁头及控制电路组成,信息存储在磁盘上,由磁头负责读出或写入。磁盘机加电后,其磁盘片就开始高速旋转(所谓 5 400 转或 7 200 转就是指盘片的转动速度)。磁头采用轻质薄膜部件,盘片在高转速下产生的气流浮力迫使磁头离开盘面悬浮在盘片上方,浮力与磁头座架上弹簧的反向弹力使得磁头保持平衡。当硬盘接到一个系统读取数据指令后,磁头根据给出的地址,首先按磁道号产生驱动信号进行定位,然后再通过盘片的转动找到具体的扇区(所耗费的总时间即为寻道时间),最后由磁头读取指定位置的信息并传送到硬盘自带的 Cache 中(此过程的数据传输速度即为硬盘的内部传输速度)。在 Cache 中的数据可以通过硬盘接口与外界进行数据交换(这时的数据传输速度就是硬盘标称的传输速度,如 UDMA66)。

3. 硬盘上信息的存储格式

磁盘被划分为若干级别的管理单位,它们分别是记录面,柱面和扇区。

硬盘一般由多个盘片组成,盘片的上下两面都能记录信息。通常把磁盘片表面称为记录面。磁盘面的面数与磁头数量是一样的。一般就用磁头号(Head)来代替记录面号。

记录面上一系列同心圆称为磁道。每个盘片表面通常有几十到上千个磁道,每个磁道又分为若干个扇区。磁道的编址是从外向内依次编号,最外一个同心圆叫 0 磁道。所有记录面上同一编号的磁道就构成了柱面(Cylinder),柱面数就等同于每个盘面上的磁道数。

每一个磁道被划分为若干个扇区(Sector)。扇区的编号有多种方法,可以连续编号,也可间隔编号。磁盘记录面经这样编址后,就可用 n 磁道 m 扇区的磁盘地址找到实际磁盘上与之相对应的记录区。对活动头磁盘组来说,磁盘地址是由磁头号、磁道号和扇区号三部分组成。

在磁道上,信息是按扇区存放的,每个扇区一般为 512 B。为进行读/写操作,就必须定出磁道的起始位置,这个起始位置称为“索引”。索引标志在传感器检索下可产生脉冲信号,再通过磁盘控制器处理,便可定出磁道起始位置。

读/写操作是以扇区为单位逐位串行读出或写入的。每一个扇区记录一个记录块。数据在磁盘上的记录格式如图 6.49 所示。

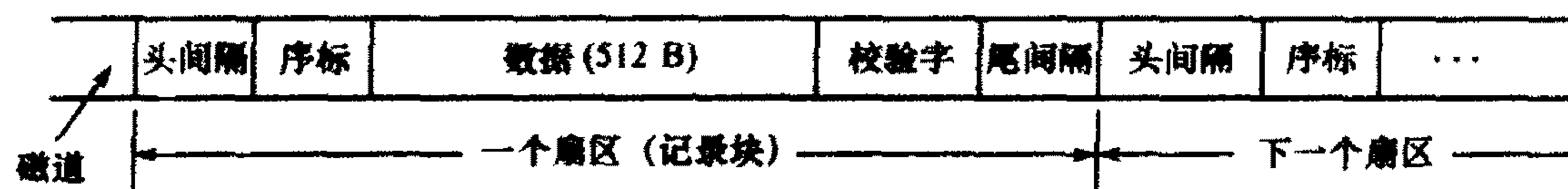


图 6.49 硬盘记录格式

图中间隔段用来留出一定的时间作为磁盘控制器的读/写准备时间,序标被用来作为磁盘控制器的同步定时信号。序标之后即为本扇区所记录的数据。数据之后是校验字,它用来校验磁盘读出的数据是否正确。

一般来说,如果知道了一个硬盘的柱面数、磁头数和扇区数,就可以知道该硬盘的存储容量(每个扇区的字节数为 512 B)。例如,柱面数为 4 096、磁头数为 16、扇区数为 63 的硬盘容量为 $16 \times 4\,096 \times 63 \times 512 = 2.1\text{ GB}$ 。

4. 硬盘和主机的数据传送方式

硬盘和主机的数据传送方式主要有 PIO (Programmed I/O) 模式和 DMA (Direct Memory Access) 模式两种。

1) PIO 模式

早期的硬盘与主机之间均以 PIO 模式进行数据传送。PIO 模式是指通过 CPU 执行程序,用 I/O 指令来完成数据的传送。由于完全用软件来控制,所以灵活性非常好,可以精细

地控制数据传送中的每一个细节问题。其缺点是数据传送速度无法提高,PIO模式的传输速率为8 MB/s~16 MB/s。

2) DMA 模式

由于硬盘容量的增大和读/写速度的提高,必然要求硬盘接口有更高的传输率。于是新的硬盘接口标准—Ultra DMA便应运而生。它一开始设计的传输速率为33.3 MB/s,是EIDE的2倍。它所采用的数据传输方式与PIO有所不同,在PIO方式中,CPU直接进行读/写控制,而Ultra DMA以突发模式在硬盘与内存之间直接进行数据传输。此外,它采用总线主控方式,由DMA控制器控制硬盘的读/写,因此节省了宝贵的CPU资源。

DMA模式在数据传送过程中不通过CPU而直接在外设与内存之间完成数据传送。惟一需要CPU介入的是进行数据传送前的初始化工作,一旦数据传送开始后,就无需CPU的干预。目前常用的DMA模式有Ultra DMA33(简称UDMA33)、UDMA66和UDMA100三种,它们的传输速率分别是33 MB/s、66 MB/s和100 MB/s。

5. 硬盘与主机的接口标准

最常用的硬盘接口标准有ATA(也称IDE或EIDE)和SCSI两种,它们定义了外存储器(如硬盘、光盘等)和主机的物理接口。

EIDE接口多用于普通微型计算机中,大多数主板都内置有一个或两个EIDE接口,用户只需将硬盘数据线插到EIDE接口即可。每个EIDE接口支持在同一数据线上连接两台设备,一个主设备和一个从设备。

SCSI接口采用了总线主控技术,即由SCSI接口来控制数据传送操作以减少CPU的负荷,这使SCSI接口的性能比EIDE接口提高了很多(但新的EIDE设备也已采用同样的技术)。SCSI接口有多种类型,每种类型的数据传输速率均不相同,常用的数据传输速率为20 MB/s、40 MB/s、80 MB/s和160 MB/s。SCSI接口的设备连接能力非常强,一个SCSI接口可以连接多达15台外部设备。

SCSI接口多用于网络服务器和工作站系统。SCSI接口的缺点是构造复杂,成本较高。

6. 磁盘阵列(Disk Array)技术

在网络系统中,服务器是一个关键性的设备。许多关键数据都存储在服务器的大容量硬盘中。一旦硬盘损坏,就会造成数据丢失,给企业造成严重的损失。与此同时,存储在服务器上的数据量越来越大,使得对服务器的硬盘容量要求也越来越高。这些都要求服务器的硬盘容量更大、可靠性更高,磁盘阵列就是为满足这种需求出现的一种数据存储技术。

磁盘阵列的全名为廉价磁盘冗余阵列(Redundant Array of Inexpensive Disk, RAID),属于超大容量的外存储器子系统。它是由许多台磁盘机按一定规则(如分条、分块、交叉存取等)组合在一起构成的。通过阵列控制器的控制和管理,磁盘阵列系统能够将几个、几十个甚至

几百个盘组合起来,使其容量高达几百 GB 至上千 GB。RAID 技术有多种实现方式,通常采用的有 RAID0(带区)、RAID1(镜像)、RAID5(奇偶校验)三种。

RAID0 是以带区形式在两个或多个物理磁盘上存储数据的卷(卷是操作系统管理硬盘空间的一种逻辑结构,它可以占据一个硬盘的部分区域,也可以占据整个硬盘或跨越多个硬盘)。带区卷不提供容错。如果带区卷上的磁盘失败,则整个卷上的数据都将丢失。

RAID1 就是将数据同时写到两个磁盘上。这两个磁盘互为镜像。如果镜像盘组中一个物理磁盘出现故障,系统可以使用未受影响的另一个磁盘继续操作。

RAID5 是具有数据和奇偶校验的容错卷,有时分布于三个或更多的物理磁盘。奇偶校验是用于在失败后重建数据的计算值。如果物理磁盘的某一部分出现错误,就可以用其他卷上的数据和奇偶校验重新创建磁盘上出错的数据。RAID5 卷也称为带奇偶校验的带区集。

6.8.2 软盘及软盘驱动器

软盘驱动器是微型计算机系统中的一个重要部件。常见的规格为 3.5 英寸,用于读/写 3.5 英寸/1.44 MB 软盘。现在微型计算机的标准配置中通常只配有 3.5 英寸软驱。软驱的数据传输率为 500 KB/s,主轴转速为 300 rpm 或是 360 rpm,道对道存取时间为 3 ms,平均存取时间为 90 ms 左右。

软盘属于可移动介质。用于 3.5 英寸软驱的软盘片叫做 3.5 英寸软盘。它分为低密度和高密度两种。低密度软盘格式化后的容量为 720 KB,盘片两面都可记录信息,每面 80 个磁道。高密度软盘每面也是 80 个磁道,但每个磁道有 18 个扇区,因此整个磁盘格式化后的容量达到 1.44 MB($2 \times 80 \times 18 \times 512 = 1.44 \text{ MB}$)。

1. 软盘的结构和记录格式

新购买的软盘必须经过格式化后方可使用。所谓格式化是指对磁盘按标准格式划分磁道、扇区,而且每个扇区按其格式填写地址信息及定义其容纳的字节数。

每张盘格式化后的容量计算公式如下:

$$\text{容量} = \text{记录面数} \times \text{磁道数/面} \times \text{扇区数/道} \times \text{字节数/扇区}$$

2. 软盘驱动器的结构

不同驱动器的内部电路不尽相同,但其逻辑结构是一致的。主要由软盘驱动机构、盘片定位机构、软盘驱动装置(通过同步电机经过皮带传动装置)、控制磁头寻道定位部件和状态检测部件:零磁道检测装置、写保护检测装置、索引孔检测装置等部件构成。

3. 读、写电路

软盘读、写数据是由同一个磁头,通过读、写电路完成的。

1) 读出放大电路,软盘记录数据是通过写电流磁化后,在磁层上保留了剩磁状态作为

记录数据的。当读数据时,在磁头中产生感应电流,用读出放大电路对读出信号进行放大、滤波、鉴别和整形,即可读出存储的数据。

2) 写电路,写入的数据经接收器整形后,送到磁头的线圈上,使磁头产生磁场,将数据记录在磁盘表面。

6.8.3 光盘

随着多媒体技术及应用软件向大型化方向发展,软盘已越来越不能满足用户的需求,人们需要一种高容量、高速度、工作稳定可靠、耐用性强的媒体来取代软盘,这样就诞生了今天的 CD-ROM 和 DVD 等产品。现在,CD-ROM 已成为微型计算机的标准配置。

1. 光盘记录信息的原理

CD-ROM 盘片由 3 层组成:透明的聚碳酸脂塑料衬底和记录信息的铝反射层以及涂漆保护层。

CD-ROM 是用铝反射层上的凹坑和非凹坑来存放信息的。聚焦的激光束照射到光盘上,利用凹坑与非凹坑反射强度的差别来读出所存信息。

在 CD-ROM 上,用凹坑的前后沿表示 1,用凹坑和非凹坑的持续长度表示 0 的个数。但若二进制信息中有连续两个以上的 1 存在时,将无法进行记录。为此需要将要存储的数据进行适当的转换。

便于在 CD-ROM 光盘上记录信息的数据编码叫做 8-14 EFM 编码。它是一种将 8 位二进制数据转换成 14 位二进制数据的编码。8 位二进制数有 256 种编码,而 14 位二进制编码可以构成 16384 种编码。我们可以从这 16384 种编码中,找到这样的编码:在两个 1 之间至少有 2 个 0 而又不多于 10 个 0。16384 种编码中满足上述要求的共有 267 种编码,从中去掉不太合适的 11 种,用余下的 256 种编码与 8 位二进制数据的 256 种编码一一对应,这样就可以构成 8 位数据和 14 位 EFM 码相对应的 EFM 转换表。

在数据编码过程中,通过查找 EFM 转换表即可将 8 位数据转换成满足要求的 14 位 EFM 码。为了在两个 14 位通道码结合的位置上也满足上述编码要求,我们在两个 EFM 码之间再增加 3 位合并码(000)。例如:

8 位数据 11101000 11100010

14 位通道 00010010000010 10010001000010

加合并码 00010010000010 00010010001000010

最后,将加有合并码的通道位信息写到 CD-ROM 上,形成前面所提到的凹坑和非凹坑。

当数据从 CD-ROM 上读出时,通过上述过程的逆过程,即可获得原来的数据。

2. 光盘及光盘驱动器的技术指标

① 容量 一张 CD-ROM 盘的标准容量为 640 MB, 也有 580 MB 和 700 MB 规格的。

② 数据传输率 最早的 CD-ROM 驱动器的数据传输率是 150 KB/s, 一般把这种速率称为 1 倍速, 记为“1X”。依次比例, 目前常见的光驱有“36X”、“40X”、“50X”等。实际上, 用户更关心的是光驱的读盘能力, 即它的纠错能力。

③ 缓冲存储器 缓冲存储器在光驱中内置的 RAM 存储器, 它用来暂存 CD-ROM 中读出的数据, 以便能够保持以恒定的数据传输率向主机传送数据。CD-ROM 驱动器中缓冲存储器的容量一般为 64 KB, 最大可达 256 KB。

④ 读取时间 读取时间是指 CD-ROM 驱动器接收到命令后, 移动光学头到指定位置, 并把第一个数据读入驱动器缓存的过程所花费的时间。目前, CD-ROM 驱动器的读取时间一般在 200 ~ 400 ms。

3. 光盘的种类

目前我们所见到的光盘从读/写方式上讲, 主要有以下几种: 1) 只读光盘(CD-ROM, CD, VCD, LD)是一次成型的产品, 由一种称作母盘的原盘压制而成, 一只张母盘可以压制数千张光盘。其最大特点是盘上信息一次制成, 可以复读而不能再写。这种光盘的数据存储量一般在 650 MB ~ 700 MB 左右; 2) DVD 数字视盘(Digital Video Disc, Digital Versatile Disk)是 1996 年底推出的新一代光盘标准, 主要用于存储视频图像。单个盘片上能存放 4.7 ~ 17.7GB 的数据。目前其最大传输速率是 2 MB/s 左右; 3) 一次性刻录光盘 CD-R, 只能写入一次的光盘。它需要用专门的光盘刻录机将信息写入, 刻录好的光盘不允许再次更改。这种光盘的容量一般为 650 MB; 4) 可擦写的光盘(MO, PD, CD-RW)与 CD-ROM 光盘本质的区别是可以重复读/写。也就是说, 对于存储在光盘上的信息, 可以根据操作者的需要而自由更改、读出、拷贝、删除。MO 光盘、PD 光盘、CD-RW 均属此类。这里仅简单介绍一下 MO 磁光盘。

MO 的全名是 Magneto Optical Disk, 是光学与磁学结合而成的一种存储技术, 当信息要写入 MO 盘的时候, 激光会聚焦在指定的位置而产生大约 300 度的高热, MO 光盘机的磁鼓便会根据写入的信息而把 MO 碟的磁性物质磁化; MO 光盘机利用另一激光去阅读 MO 光盘上磁化物质的极性, 反射回来的激光会根据不同的偏振角度得出所存储的信息。MO 光盘可以不限次数地读/写。市面上流行的 3.5 寸 MO 光盘容量有 230 MB、640 MB 和 1.5 GB 三种。

6.8.4 可移动外存储器(USB 硬盘)

1. 闪存盘(优盘)

由前面的章节介绍到, 可以使用闪速存储器(Flash Memory)做移动存储媒介来取代低性

能的软盘,简称内存,又称为 USB 硬盘,如图 6.51 所示。在目前已成为流行趋势。这种类型的移动存储器用通用串行总线(USB)接口与主机相连,可以像使用软、硬盘一样在该盘上读/写、传送文件。目前的 Flash Memory 产品可擦写次数都在 100 万次以上,数据至少可保存 10 年,而存取速度至少比软盘快 15 倍以上。USB Flash Disk 由多种容量的选择,从 16 MB ~ 1 GB,未来可达 2 GB 以上,闪存的可可靠性远高于磁盘,对于数据安全性提供了更好的保障。工作时不需要外接电源,可热拔插,且因其体积小,能抗震防潮,耐高低温,很适于携带。Windows ME 以上的操作系统不需要安装驱动程序。对于 BIOS 支持 USB - ZIP 的主板,USB Flash Disk 还提供了系统引导功能。只要执行 Windows 98 SE 驱动程序中的应用程序“mFormat”,并且选中“制作启动盘”选项,即开始制作启动盘,同时,USB Flash Disk 内的数据也不会被删除。如果 USB Flash Disk 数据出错而不能工作时,重新格式化即可使用。



图 6.51 USB 硬盘

2. USB 硬盘盒

如果有一个带有 USB 1.1 接口标准的硬盘盒和一个小巧的笔记本电脑专用移动式硬盘,再加一根 USB 接口线,用户也可以自制一块即插即用的 USB 硬盘,事实上,现在市面上大量的 10 GB ~ 20 GB 之间的 USB 硬盘即是这种类型。这类 USB 硬盘的使用方法与优盘相同。目前 USB 接口的数据传输速度为 12 MB/S。比硬盘速度低,但与软盘相比后者还要快得多。

习 题 六

6.1 计算机的存储器是用来存放(),随机访问存储器的访问速度与()无关。计算机系统中的存储器分为()和()。在 CPU 执行程序时,必须将指令存放在()中。半导体存储器主要分为只读存储器、()、()等。CPU 的寻址能力最基本的因素取决于()。

6.2 什么是 Cache? Cache 能够极大地提高计算机的处理能力是基于什么原理?

6.3 RAM 为什么需要刷新? 什么是最大刷新周期?

6.4 如何解决 Cache 与主存内容一致性问题?

6.5 新购买或擦除的 EPROM,各单元的内容是什么?

6.6 要组成容量为 $4\text{ M} \times 8\text{ bit}$ 的存储器,需要 $4\text{ M} \times 1\text{ bit}$ 的存储器多少片? 或者需要 $1\text{ M} \times 8\text{ bit}$ 的存储器多少片?

6.7 欲设计具有 $64\text{ K} \times 2\text{ bit}$ 存储容量的芯片,如何安排地址线和数据线的数目,才能使两者之和最小?

6.8 用 $8\text{ K} \times 8\text{ bit}$ 的 ROM 芯片和 $8\text{ K} \times 4\text{ bit}$ 的 RAM 芯片组成存储器,按字节编址,其中 RAM 的地址为 $0000\text{H} \sim 3\text{FFFH}$,ROM 地址为 $4000\text{H} \sim 7\text{FFFH}$,画出此存储器组成结构图及与 CPU 的连接。

6.9 某页式管理虚拟存储器按 1 KB 分页,即每页大小为 1 024 B ,页表的一部分如下,问当访问虚地址 1 023 单元、 2 049 单元和 4 095 单元时,其对应的实地址各为多少?

虚页号	实页号
0	2
1	1
2	4
5	0

6.10 从右图中判断 8088 系统中存储系统译码器 74LS138 的输出 $\overline{Y_0}$ 、 $\overline{Y_4}$ 、 $\overline{Y_6}$ 和 $\overline{Y_7}$ 所决定的内存地址范围。

6.11 与逻辑地址 $2\text{FAEH:}4320\text{H}$ 项对应的物理地址是什么?

6.12 若主存 DRAM 的存取周期为 70 ns ,Cache 的存取周期为 50 ns ,则由它们构成的存储系统的平均存取周期是多少?

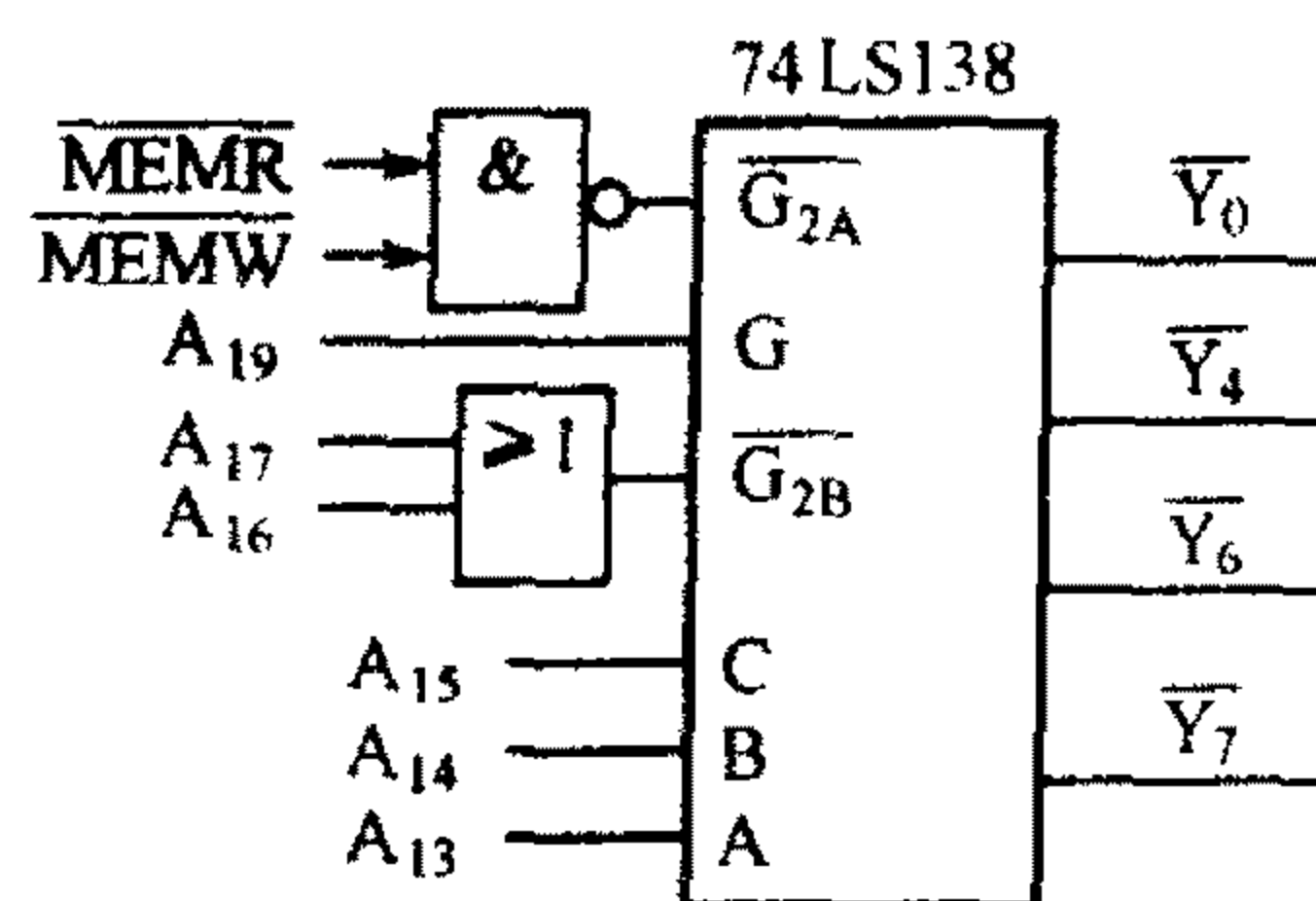
6.13 说明 28F040 的编程过程。

6.14 光盘的主要技术指标有哪些?

6.15 Windows 存储管理规则,每页的字节数是多少?

6.16 某以 8088 为 CPU 的微型计算机内存 RAM 区为 $00000\text{H} \sim 3\text{FFFFH}$,若使用 $6264(8\text{ K} \times 8\text{ bit})$ 和 $2164(64\text{ K} \times 1\text{ bit})$,各需要多少芯片?

6.17 利用全地址译码将 6264 接在 8088 的系统总线上,其所占地址范围为 $\text{A}0000\text{H} \sim \text{A}1\text{FFFH}$,画出连接图。



第7章 输入/输出技术

本章讨论接口的一般概念和功能、微型计算机系统中的几种基本输入/输出方法以及中断技术,并介绍常用中断控制器芯片 8259 的特性及应用。

7.1 输入/输出系统概述

微型计算机的硬件系统是由主机和外部设备两部分组成的。微处理器在运行中所需要的程序和数据要由外部设备输入,而处理的结果则要输出到外部设备中去。控制并实现信息输入/输出的就是输入/输出系统(Input Output System),它提供了处理器与外部世界进行信息交换的各种手段。这里的外部世界包括了提供数据输入/输出的设备、辅助存储器、操作控制台、其他处理器、各种通信设备以及使用系统的人等,这些设备相对处理器来讲,无非是将信息输入或将结果输出。因此,把除人以外的各种设备都统称为输入/输出设备或外部设备。

在计算机系统中,一般将除 CPU 和主存储器之外的部分称为输入/输出系统,包括输入/输出设备、输入/输出接口和相应的输入/输出软件。

输入/输出设备的种类繁多,如常见的键盘、鼠标、扫描仪、显示器、打印机、绘图仪等。它们在功能、品种、技术指标等多方面都有各自的特性。既有电子式的,也有电动式甚至机械式的;既可以是数字信号,也可以是模拟信号。因此,输入/输出系统是计算机系统中最具有多样性和复杂性的部分,也是极其重要的环节之一。

7.1.1 输入/输出系统的特点

外部世界的多样性和复杂性,使得输入/输出系统具有以下 4 个方面的特点:

1) 复杂性

输入/输出系统的复杂性表现在两个方面,一是输入/输出设备的复杂性。这点已由上面的叙述可知,这种复杂性对用户来讲大多都隐藏在操作系统中,由操作系统统一管理,并为用户提供方便、友好的使用界面,用户则无需对输入/输出设备的具体工作细节进行了解。

复杂性表现的另一个方面是处理器本身和操作系统所产生的许多随机事件都需由输入/输出系统来处理,如中断等。

2) 异步性

CPU 的各种操作都是在统一的时钟信号作用下完成的,各种操作都有自己的总线周期,而不同的外部设备也有各自不同的定时与控制逻辑,且大都与 CPU 时序不一致,它们与 CPU 的工作通常都是异步进行的。当某个输入设备有准备好的数据需要向 CPU 传送或输出设备的数据寄存器空可以接收数据时,一般要先向 CPU 提出服务请求,如果 CPU 响应请求,就转去执行相应的服务。对 CPU 来讲,这种请求可能是随机的,每两次这样的请求之间可能间隔很短,也可能相隔时间较长,而且在响应请求之前,外设可能已为“准备好”运行了相当一段时间。如此,输入/输出系统相对于 CPU 就存在操作上的异步性和时间上的任意性。

一般来讲,一个微处理器要管理多台外部设备,要求在任意两次 CPU 与外设交往的时刻之间,CPU 要能够全速运行它自己的程序,或管理其他外部设备,以保证 CPU 与外设之间及各外部设备自身之间都能并行工作,不必相互等待,从而提高整个系统的效率。要实现这一点,需要采用本章下面将要讨论的中断输入/输出方式或直接存储器存取(DMA)方式。

3) 实时性

用做实时控制系统的计算机对时间的要求很高。实时性是指处理器对每一个连接到它的外设或处理器本身,在需要或出现异常时,如电源故障、运算溢出、非法指令等,都要能够给予及时的处理,以防止错过服务时机使数据丢失或产生错误。外部设备的种类很多,信息的传送速率相差也很大,如有的是单字符传送,即每次只传送一个字符,像打印机等,传送速度为每秒几个到几十个字符;而有的则是按数据块或按文件传送,像磁盘等,每秒传送几到几十兆字符。因此,要求输入/输出系统能够保证处理器对不同设备提出的请求都能提供及时的服务,这就是输入/输出系统的实时性要求。

4) 与设备无关性

由于输入/输出设备在信号电平、信号形式、信息格式及时序等多方面的差异,使得它们与 CPU 之间不能够直接地连接,而必须通过一个中间环节,这就是输入/输出接口(Input Output Interface,简称 I/O 接口)。为了适应与不同外设的连接,规定了一些独立于具体设备的标准接口,如串行接口、并行接口等。不同的型号外设可根据自己的特点和要求,选择一种标准接口与处理器相连。对连接到同一种接口上的外设,它们之间的差异由设备本身的控制器通过软件和硬件来填补。这样,CPU 能够通过统一的软件和硬件来管理各种各样的外部设备,而不需要了解各种外设的具体细节。例如,在 Windows 9X 操作系统中,凡经过 Microsoft 公司测试过的机型和外设都可直接相连,由操作系统统一进行管理。

7.1.2 输入/输出接口的基本功能

一个输入/输出系统简单地讲至少应具备以下两项基本的功能:

- ① 为数据的传送操作选择输入/输出设备;

② 在选定的输入/输出设备和主机之间交换数据。

系统的这两项功能由操作系统和输入/输出接口共同完成。在这里,操作系统负责输入/输出操作的管理,并提供简单的使用界面;处理器与外部设备之间的连接和信息的传送则是通过 I/O 接口进行,在这里,I/O 接口起着连接外部设备和处理器的“桥梁”的作用。它的一部分负责和计算机系统总线连接,另一部分则负责和外部设备连接,其示意图如图 7.1 所示。

负责与系统总线连接的部分主要包括:数据信号线、控制信号线和地址信号线。数据线除实现数据的接收和发送外,还负责传送 CPU 发给接口的编程命令及接口送出的状态信息;控制信号主要是读/写控制信号;地址信号一般通过译码电路连接到接口的片选引出线上,从而确定接口所占的地址或地址范围。

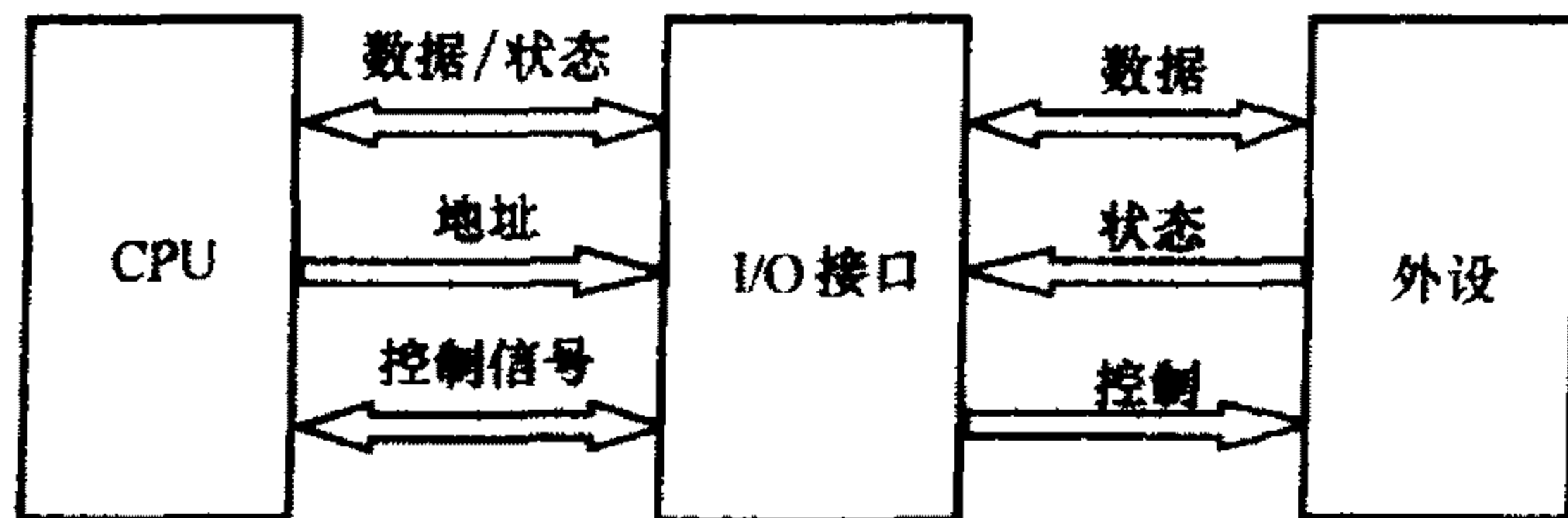


图 7.1 主机与外设通过接口连接

由于处理器与外部设备之间在工作速度、信号形式、信息格式、工作时序等多方面都存在较大的差异,要使输入/输出设备与处理器之间实现正常的信息传送,I/O 接口必须具备以下几项基本功能:

① I/O 地址译码与设备选择 所有外部设备都是通过 I/O 接口挂接在系统总线上的,地址译码功能可使得在任一时刻只允许一个外设与 CPU 通过总线进行数据传送,而其他未被选中的 I/O 接口则呈现高阻状态,与总线隔离。

② 信息的输入/输出 CPU 通过 I/O 接口与外部设备进行信息交换,并可通过向接口写入命令的方式控制、监测与管理 I/O 接口和外设的工作状态。另外,外设也通过接口向 CPU 发出中断请求。

③ 命令、数据和状态的缓冲与锁存 由于 CPU 的速度很高,而外设的速度有高有低,而且不同的外部设备速度差异很大,从而使得 CPU 与外设之间在工作时序和速度上有很大差异。为了确保计算机和外设之间可靠地进行信息传送,要求接口电路应具有信息缓冲存储能力。不仅要缓存 CPU 传送给外部设备的信息,以使 CPU 能及时地执行自身的程序,还要缓存外设传送给 CPU 的信息,以实现 CPU 与外设之间信息的同步交换。

④ 信息转换,包括信号形式、信息格式、时序等的匹配 CPU 只能处理数字信号,而外

设的信号形式多种多样,有数字量、开关量、模拟量(电流、电压、频率、相位),甚至还有非电量,如压力、流量、温度、速度等。因此,I/O 接口还要实现信息格式的转换、电平转换、码制转换、传送管理以及联络控制等功能。

7.1.3 I/O 端口

1. I/O 端口的编址方式

CPU 通过 I/O 接口进行信息传送,是将数据输入到接口内部的数据寄存器,再由外设将数据读走;或是在外设将数据放入接口的数据寄存器后,CPU 再从该寄存器将数据取走。接口的内部有一组寄存器,包括存放数据的数据寄存器、存放状态信息的状态寄存器以及用于命令传送的控制寄存器。把接口内部的这些寄存器称为 I/O 端口(I/O Port),相应的有:数据端口、状态端口和命令(或控制)端口。根据需要,一个 I/O 接口可能仅包含其中的一类或两类端口,也可能包含全部三类端口。而一个外部设备可以对应一个或多个端口,所以有时也将端口地址称为外设地址。处理器在与外设通信时,通过数据端口从外设读入数据(或向外设输出数据),从状态端口读入设备的当前状态,通过命令(控制)端口向外设发出控制命令。

由于处理器可能管理多个端口,为了区分这些端口,从而使 CPU 方便地实现对 I/O 设备的寻址和选择,就必须给每一个端口像内存单元那样分配一个惟一的地址,该地址也称为外设地址或 I/O 地址。在微型计算机系统中,端口的编址通常有两种不同的方式。一是与内存单元统一编址,二是独立编址。

1) I/O 端口与内存单元统一编址

这种编址方式又称为存储器映射编址方式,即把每个 I/O 端口都当做一个存储单元看待,I/O 端口与存储器单元在同一个地址空间中进行编址。通常是在整个地址空间中划分出一小块连续的地址分配给 I/O 端口。被端口占用了的地址存储器不能再使用。图 7.2 为 I/O 端口与内存单元统一编址的示意图。图中,分配给 I/O 端口的地址范围为 F0000H ~ FFFFFH,共 65 536 个地址。

统一编址方式的优点是可以访问内存的方法来访问 I/O 端口。由于访问内存的指令种类丰富、寻址方式多样,因此这种编址方式为访问外设带来了很大的灵活性。从原则上讲,第 3 章中介绍的所有用于内存的指令都可以用于外设,不需再设专门的 I/O 指令。同时,I/O 控制信号也可与存储器的控制信号共用,这也给应用带来了很大的方便。

统一编址方式的缺点是外设占用了一部分地址空间,这就减少了内存可用的地址范围,

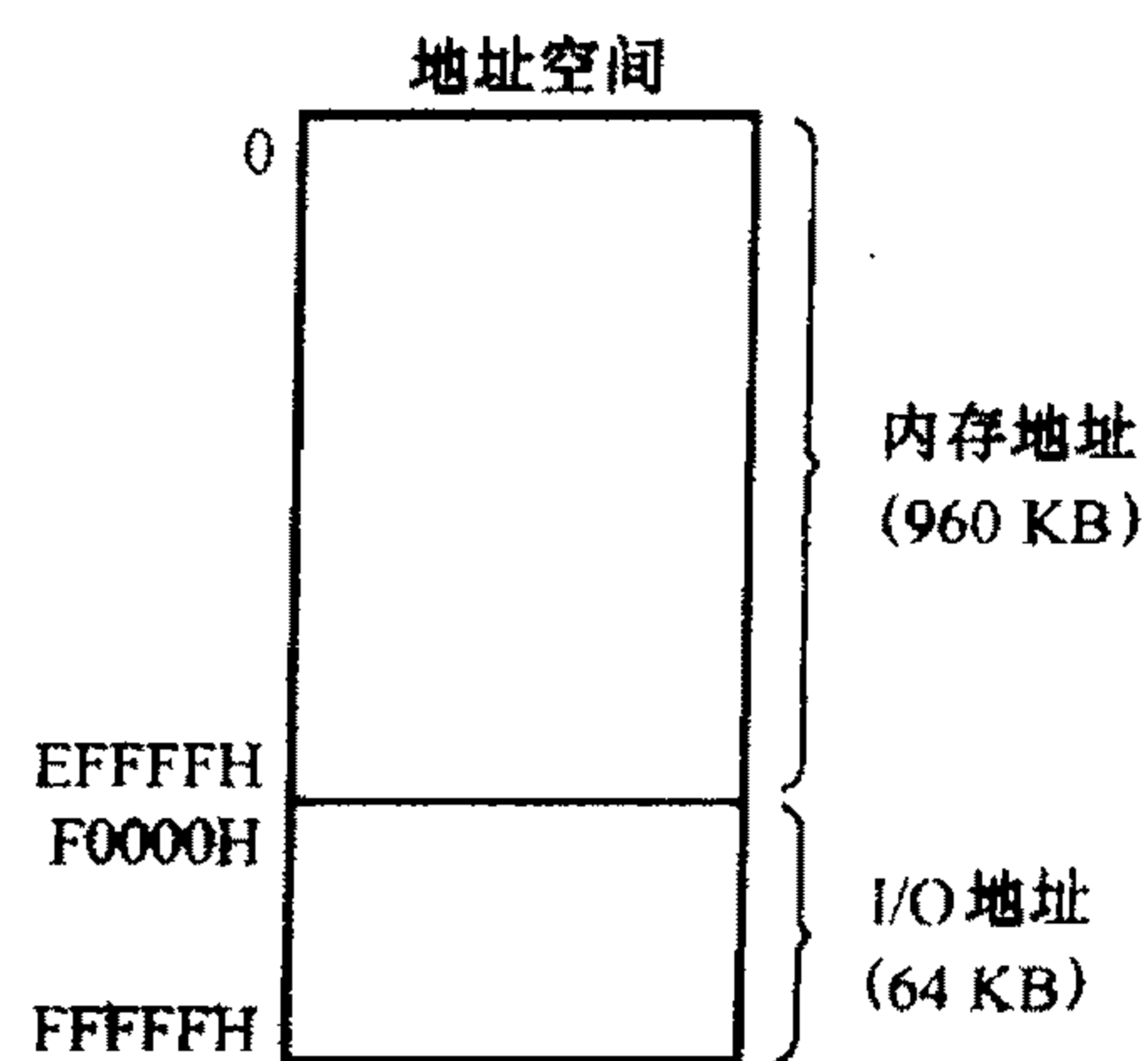


图 7.2 I/O 端口与内存统一编址方式示意图

因此对内存容量有潜在的影响。此外,从指令上不易区分当前是对内存进行操作还是对外设进行操作。

2) I/O 端口独立编址

I/O 端口独立编址时,内存地址空间和外设地址空间是相互独立的,系统设有专门的 I/O 操作指令(IN 和 OUT),由指令的操作数部分直接或间接地给出输入/输出端口地址。这种独立编址方式在 Z80 系列及 Intel 公司的 80X86 系列 CPU 中得到广泛采用。8086/8088CPU 就采用了 I/O 端口独立编址方式。例如,8086CPU 使用地址总线的低 16 位作为寻址 I/O 端口的地址线(此时地址总线的高 4 位为 0),故 8086 最多可以寻址 64K 个 8 位地址码的端口,或 32K 个 16 位地址码的端口。80386 的地址总线是 32 位,可以直接访问 4 GB 的存储空间或 I/O 空间。对 80386 以上的 CPU,I/O 端口的地址码可以是 32 位,指令的执行是将指定的端口地址中的字送 AX,将端口地址 + 2 中的高 16 位字送 EAX 的高 16 位。

8086CPU 的地址总线宽度为 16 位,系统可寻址的内存地址范围为 00000H ~ FFFFFH,而外设端口的地址范围为 0000H ~ FFFFH,这两个地址空间相互独立,互不影响,CPU 使用不同的控制信号来区分是对内存操作还是对 I/O 端口操作。

由 8086CPU 组成的系统可以工作在最大和最小两种模式下。在不同的工作模式下,8086 区分对端口和内存操作的控制信号是不一样的。

① 在最小模式下,CPU 的读命令 RD 和写命令 WR 对存储器和 I/O 端口的访问是公用的。当内存地址码和端口地址码不重合时,由 M/\overline{IO} 信号来区分是对内存操作还是对端口操作。当 $M/\overline{IO} = 1$ 时,表示访问存储器,由此地址总线上给出的是某个存储单元的地址;当 $M/\overline{IO} = 0$ 时,则表示 CPU 执行的是 I/O 的读/写操作,这时地址总线上给出的是某个 I/O 端口的地址。

② 在最大模式下,CPU 的控制信号通过总线控制器 8288 连接到控制总线上,由总线控制器分别产生存储器读/写($\overline{MRDC}/\overline{MWTC}$)命令及 I/O 端口的读/写($\overline{IORC}/\overline{IOWC}$)命令,用它们来代替公用的 M/\overline{IO} 信号。

综上所述,I/O 端口独立编址的特点是:

- ① I/O 端口的地址空间与内存地址空间完全独立;
- ② I/O 端口与内存使用不同的控制信号;
- ③ 指令系统中设置了专门用于访问外设的 I/O 指令。

大多数的 8 位接口电路芯片都包含有多个端口。为管理方便,通常将各端口的地址分配在一个连续的地址块,这个地址块中最小的那个地址称为(外设的)基地址(Base Address)。在这些连续的地址中,有奇地址也有偶地址。当 8 位的接口芯片与 8 位的 CPU(如 8088)相连时,可直接连接。但在与具有 16 位数据总线的 CPU(如 8086)相连时存在是与高 8 位还是低 8 位数据总线相连的问题。理论上讲,它既可以与低 8 位连接,也可与高 8 位

连接;但低 8 位数据总线只能传送具有偶地址的 I/O 端口的数据,高 8 位数据总线则只能传送具有奇地址的 I/O 端口的数据。一般可采用两种方法来解决这一问题。

- I/O 端口只使用偶地址 此时,I/O 接口的 8 位数据线与处理器数据总线的低 8 位相连,地址总线的最低位 A_0 取低电平。若接口内包含数个端口,则端口地址都应设成偶地址(即取 $A_0 = 0$),通常可使 $A_0 = 0$ 作为该接口电路芯片的选片信号。

- I/O 端口使用连续的地址 此时须附加 8 位至 16 位数据的转换逻辑电路,如图 7.3 所示。假设图中的接口电路中包含 4 个端口地址,当 CPU 访问偶地址端口时,最低位地址信号 A_0 为低电平,地址译码器输出使得 1# 8286 被选中,接口的 8 位数据信号与 CPU 的低 8 位数据总线 $D_0 \sim D_7$ 接通。而此时由于 $\overline{BHE} = 1$,2# 8286 不能被选中,使与 CPU 的高 8 位数据线断开。而在 CPU 访问奇地址时,则 $A_0 = 1$,2# 8286 被选中,1# 8286 断开。

为便于读者理解,本书对接口电路的讨论主要针对 8 位接口芯片与 8 位 CPU 的连接。

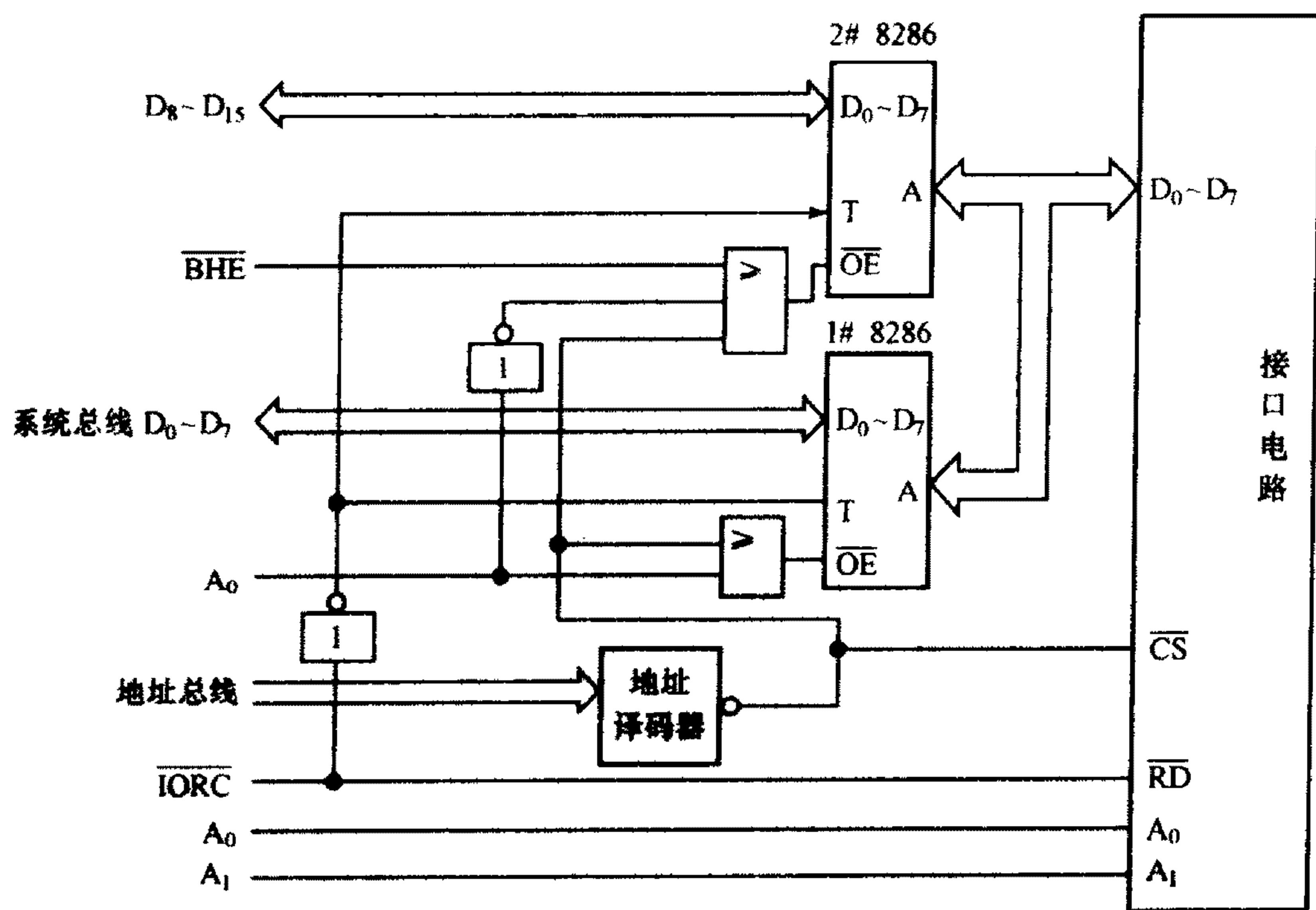


图 7.3 8 位 I/O 接口与 16 位数据总线的连接

2. 端口地址译码技术

在计算机系统中,每个 I/O 端口都要占用一个地址,CPU 在通过 I/O 指令与输入/输出接口进行信息交换时,必须要首先确定端口地址,即在执行 I/O 指令时,首先要将所要访问端口的地址放到地址总线上,这组地址信号经过一个中间环节,最后转换为一个有效的电信号(电平信号或边沿信号)接到接口芯片的“使能”(Enable)端,表示选中该芯片,然后再进行

读/写操作。这种将总线上的地址信号转换为某个接口芯片的“使能”信号的过程就称为地址译码,实现这个过程的“中间环节”叫做译码电路。

有关译码的技术在第6章已经接触到了。要使一个存储器芯片在整个存储空间中占据一定的地址范围,是通过高位地址信号的译码来确定的。在输入/输出技术中,由于一个接口芯片上可能包含一个或数个端口地址,所以也可以同存储芯片的译码类似,利用高位地址信号与CPU的控制信号一起通过译码,产生接口芯片的片选信号(即使能信号),以确定该接口在I/O端口地址中的范围,再通过低位地址信号确定芯片上各端口的地址。

I/O端口地址的译码方法一般有固定式和可选式两种。目前常用的各种I/O接口卡大多采用固定式端口地址译码。所谓固定式译码是指接口芯片上的端口地址号是固定的;而可选式译码则是指接口芯片的地址号能够根据要求改变,从而适应不同场合地址分配的需要。可选式地址译码可通过开关跳线、微型拨动开关或比较器加地址开关译码等方法实现,有关这方面的介绍读者可参阅相关书籍。本书仅讨论固定式端口地址译码技术。

固定式地址译码可通过门电路或专门译码器实现。如果一个接口芯片仅包含一个端口地址,可利用各种逻辑门电路构成地址译码电路。当接口芯片中包含多个端口地址时,这些地址通常是连续的,此时采用专门的译码器会比较方便。如使用本书第2章介绍的74LS138译码器,可实现包含8个端口地址的接口芯片的地址译码。除这种3线-8线译码器外,类似的还有4线-16线译码器74LS154、双2线-4线译码器74LS139等,这里就不再一一介绍。

相对于存储器芯片的译码,端口地址译码技术中有几点要注意:

① 8086CPU能够寻址的内存空间为1MB,其20位物理地址由地址总线的全部20位信号产生,其中高位($A_{19} \sim A_1$)用于确定存储芯片的地址范围,低位($A_0 \sim A_{15}$)用于片内寻址;而8086CPU能够寻址的I/O端口仅为64K(65535)个,即只需使用地址总线的低16位就可使每个端口拥有惟一地址,也就是说,参与译码的地址信号最多只是 $A_{15} \sim A_0$ 。对只有单一端口地址的接口,这16位地址线应全部参与译码,译码输出直接选择该端口;而对具有多个I/O地址的接口芯片,则16位地址线的高位参与译码,以决定接口的基地址(即接口所占的地址范围),而低位则用于确定要访问接口芯片上的哪一个端口。

② 当8086CPU工作在最大模式时,对存储器的读/写要求控制信号 $\overline{\text{MEMR}}$ 或 $\overline{\text{MEMW}}$ 有效;如果是对I/O端口读/写,则要求控制信号 $\overline{\text{IOR}}$ 或 $\overline{\text{IOW}}$ 有效。

③ 地址总线上呈现的信号是内存的地址还是I/O端口的地址,取决于8086CPU $\overline{\text{M}}/\overline{\text{IO}}$ 引出线的状态。当该引出线状态为1时为内存地址,即CPU正在对内存进行读/写操作;当状态为0时则为I/O端口地址,即CPU正在对I/O端口进行读/写操作。

I/O地址译码电路的构造与存储器基本相同,这些在第6章已经介绍,此处不再赘述。

7.2 常用输入/输出方法

在计算机系统中,针对具有不同工作速度、工作方式及工作性质的外部设备,可以采用不同的输入/输出方法。常见的有:

- ① 程序控制方式;
- ② 中断控制方式;
- ③ 直接存储器存取(DMA)方式;
- ④ 通道控制方式。

但对微型计算机系统来讲,目前常用的基本输入/输出方式只有上述的前三种,即程序控制、中断控制和 DMA 方式。在大型计算机系统中,另外增设了专门用于数据输入/输出的硬件装置——输入/输出通道,以提高与外部的通信能力,利用它进行数据传送的方法就称为 I/O 通道控制方式。

7.2.1 程序控制方式

程序控制方式就是通过程序来控制主机和外部设备之间的信息传送,即在程序中加入一段包括输入/输出指令在内的程序段,直接控制 CPU 与输入/输出设备之间的信息传送。

根据外部设备性质的不同,程序控制方式又可分为无条件传送方式和查询工作方式。

1. 无条件传送方式

这种数据传送方式是在程序执行输入/输出指令时,无条件地执行指令相应的操作。这种方式主要用于外部控制过程的各种动作时间是固定的而且是已知的情况,针对的是一些简单的、随时“准备好”的外设。也就是说,在这些设备工作时,随时都可以接收 CPU 输出的数据,或者它们的数据随时都可以被 CPU 读出,即 CPU 不需要查询外部设备当前的状态就可以无条件地进行数据的输入/输出。例如,输出代码使 7 段数码管显示一个相应的数据,数码管随时都可以接收这个输出的数据;又如,要想知道一个开关的状态,通过输入指令就能够随时读到开关确切的状态(打开或者闭合,如图 7.4 所示)。在与诸如此类的外部设备进行数据传送的过程中,数据交换与指令的执行是同步的,因此这种方式也可称为同步传送方式。

当 CPU 从外部设备读入数据时,执行一条 IN 指令,将低 16 位地址信号组成的端口地址送上地址总线,经过译码,选中对应的端口,然后在 $\overline{IOR} = 0$ 期间将数据读入 CPU。输出的过程类似,只是必须在 \overline{IOW} 有效时将数据写入外设。

图 7.4 所示的电路是一个典型的无条件输入的例子。开关 S 可以看做是一个简单的外设,它的状态是确定的,要么闭合,要么打开。当计算机通过三态门接口进行读操作时,就读入了开关 S 在指令执行时刻的状态。如果输入数据的 $D_0 = 0$,就表示开关 S 处于闭合状态;若 $D_0 = 1$,则开关 S 处于断开状态。

除了开关和数码管外,无条件传送方式还适用于继电器、步进电机等其他类似的外设以及从开关量或测试某些物理量的传感器发出的缓慢变化的信号(此时要求 CPU 读取数据的频率能够跟上信号的变化)。

2. 查询工作方式

查询工作方式也称为有条件传送方式或应答式传送方式。CPU 在传送数据之前,必须先检查外设的状态,若外设已做好收发数据的准备,则处理器可以进行数据的接收或发送,否则 CPU 就要等待,直到外设准备好为止。这种利用程序不断地询问外部设备的状态,并根据它们当前的状态来实现数据的输入和输出的方式就称为程序查询方式。从查询方式的工作流程可知,外部设备需向计算机提供一个状态信息,相应的接口除要具有传送数据的端口外,还应具备一个传送状态的端口。

图 7.5 是查询方式下的系统构成示意图。图中,接口与外设之间有三类信息传送,一类是要传送的数据,一类是外部设备的状态信息,最后一类是 CPU 通过接口发出的控制信号。工作中,CPU 不断查询外设的状态,判断外设是否准备好进行数据传送,必要时还需送出控制信号。

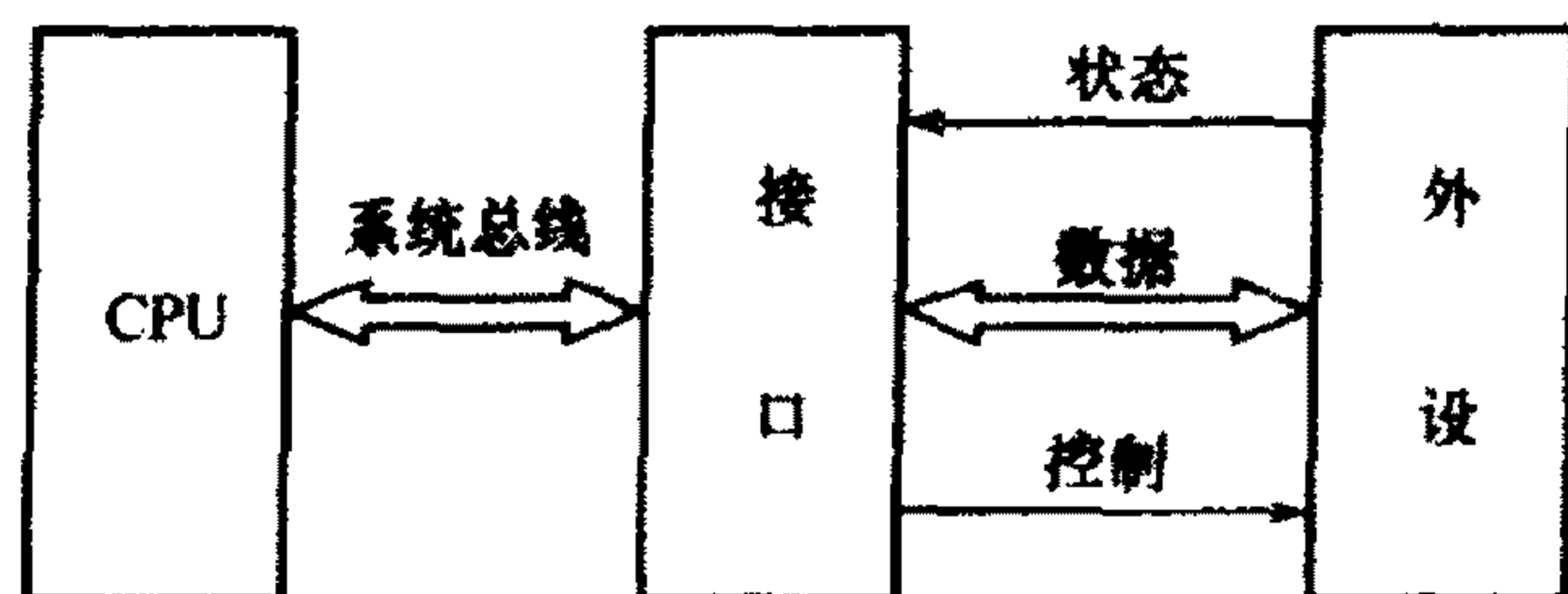


图 7.5 查询方式下的系统构成示意图

图 7.5 所示的系统中仅连接了一个外部设备。对这种单一外设采用查询方式传送数据的工作过程可分为这样几步,而每一步需要用一条或两条指令实现:

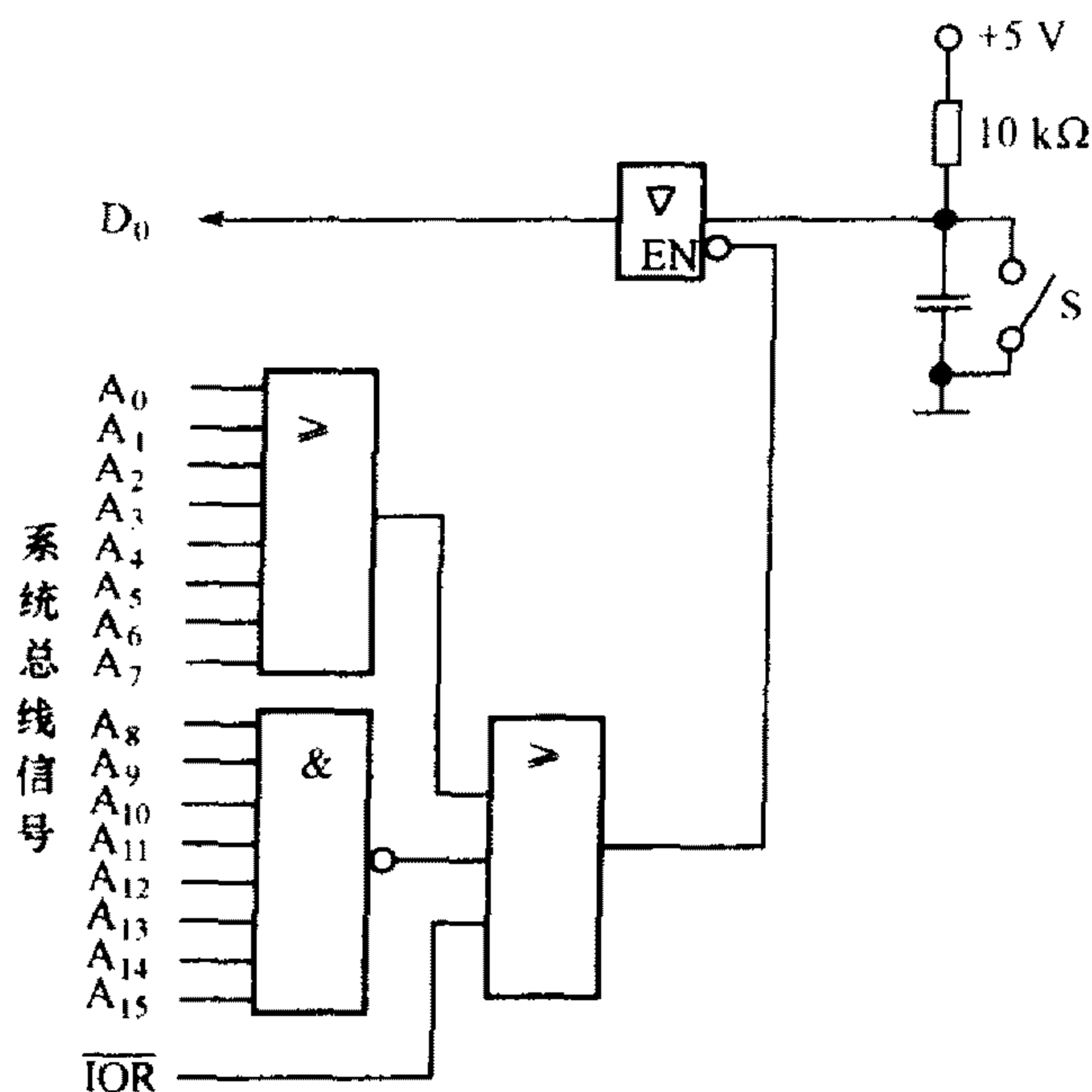


图 7.4 无条件数据传送

- ① 查询外部设备的状态 利用输入指令将外设的状态字读入 CPU 的累加器中。
- ② 检测相应的状态位 读入的状态字为 8 位或更长(视机器字长而定),但有效的状态位可能只有 1 位或 2 位等,可利用逻辑运算指令屏蔽不相干的位,再去检测状态位的状态,看外设是否处于“准备好”状态(对读操作有数据准备好,对写操作有空的接收数据寄存器)。
- ③ 如果没有准备好,则等待,重复以上两个步骤;否则 CPU 就执行预定的数据传送。
- ④ 若为读操作,则在数据读入后,CPU 向外设发响应信号,表示数据已被接收,外设收到响应信号之后,即可开始下一个数据的准备工作;若为写操作,则 CPU 在向外设送出数据的同时还发出输出就绪信号,就绪信号用来通知外设已送来有效数据,外设接收数据后,向 CPU 发出数据已收到的状态信息。
- ⑤ 在一次数据传送结束后,通过与数据长度计数器值的比较,来判断数据是否传送完毕。若没有结束,则重复以上各步骤。

上述查询方式的工作流程如图 7.6 所示。

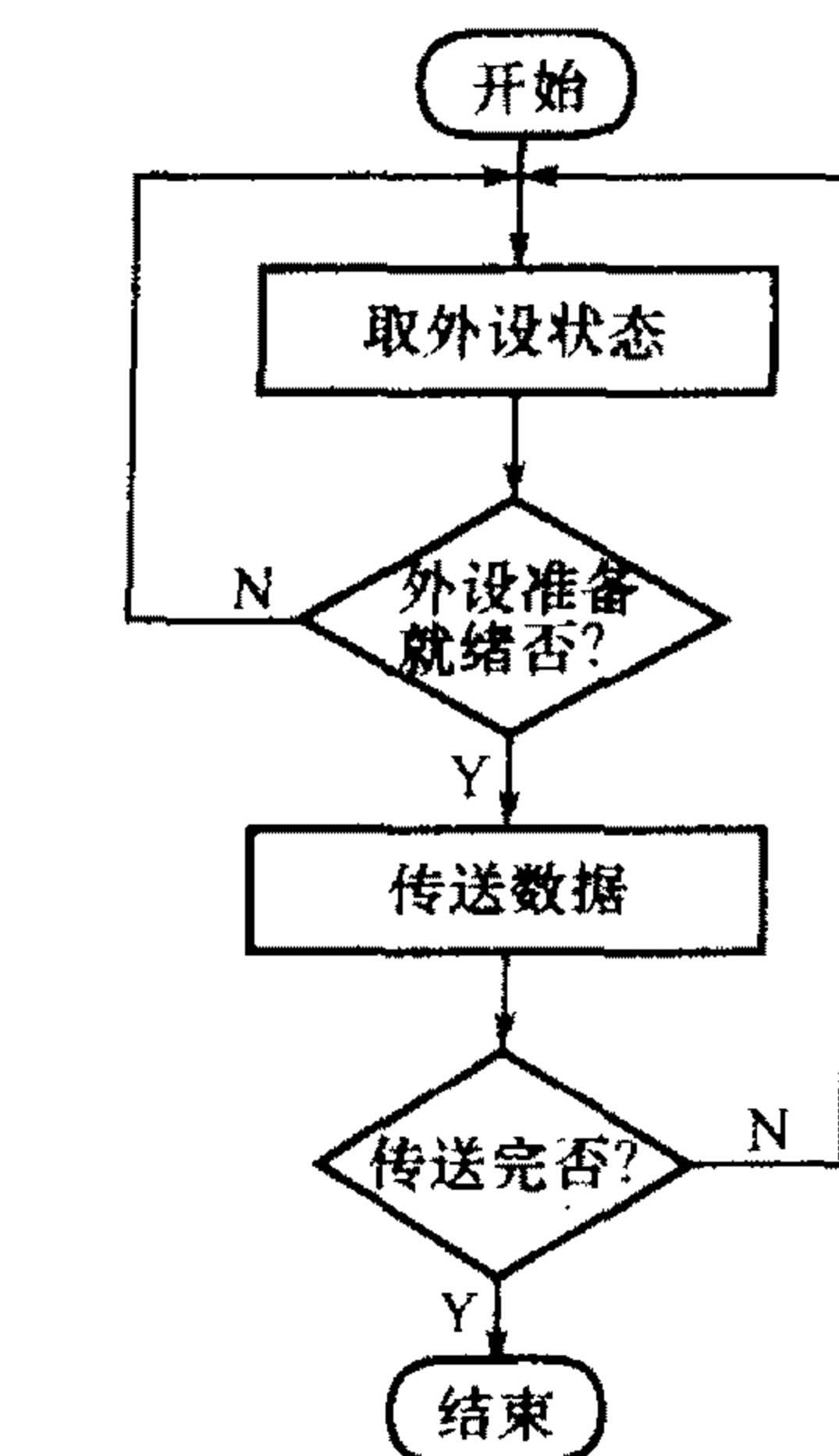


图 7.6 查询方式的工作流程

例 7-1 某外设与系统的连接如图 7.7 所示。当 $\overline{\text{BUSY}}$ 端为低电平(=0)时,表示可以向外设传送一次数据,为高电平(=1)时,则表示外设忙。现编写程序将 DATA 下 100 B 的数据块通过该接口传送到外设。

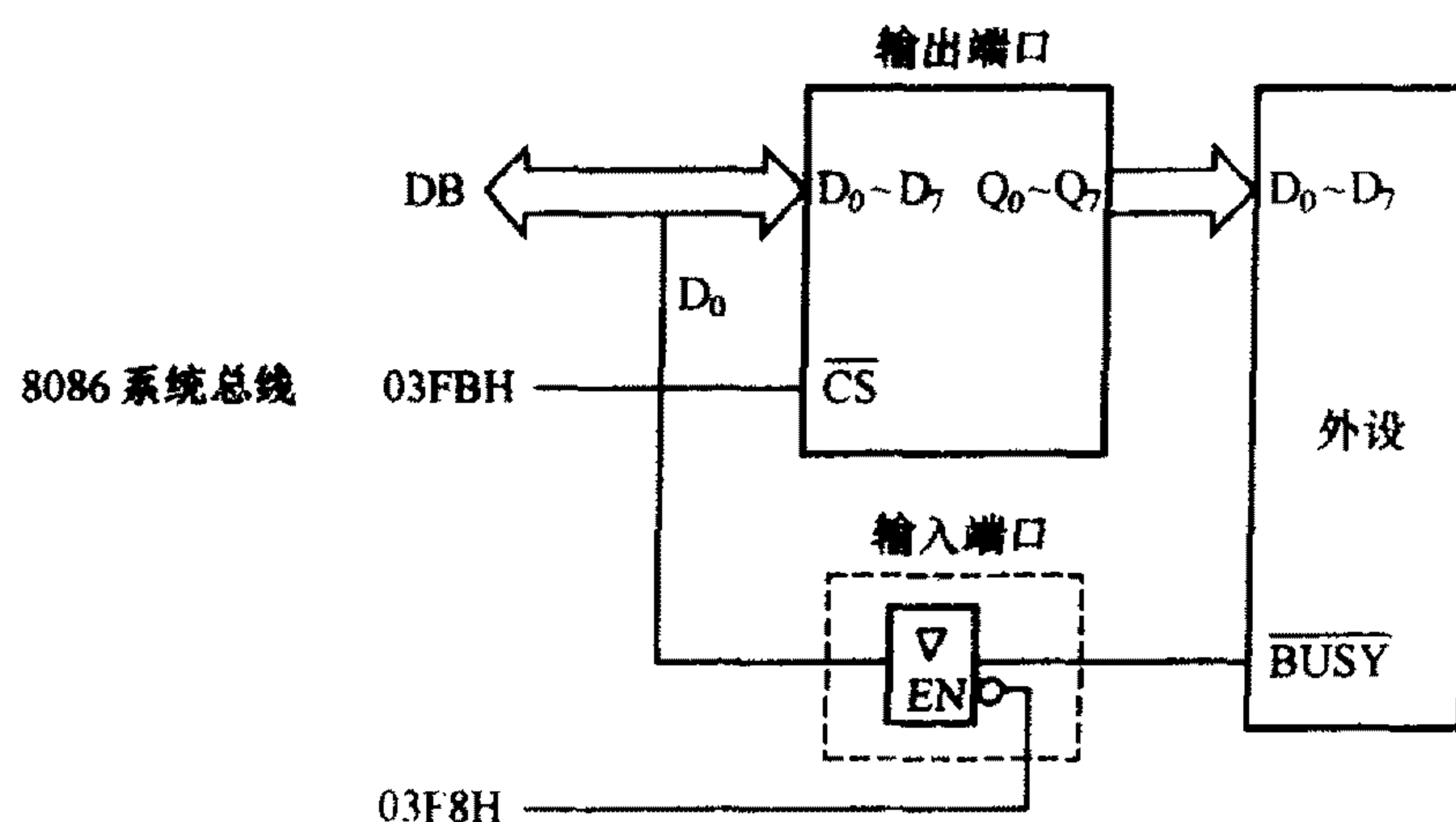


图 7.7 例 7-1 图

分析 这是一个典型的应用查询工作方式进行数据传送的例子,即通过查询外设的状态,确定是否可以进行一次数据传送。实现一次传送后,还要判断数据是否传送结束,没有结束,则重复上述过程。整个程序的编写可完全参照图 7.6 所示的流程。

由图 7.7 可知,状态输入端口的地址为 03F8H,数据输出端口地址为 03FBH,状态信号通过输入端口接到数据总线的 D₀ 端。程序如下:

```

        LEA SI, DATA      ;取数据块偏移地址
        MOV CL, 100        ;设置数据块长度
AGAIN:  MOV DX, 03F8H      ;取输入端口地址
WATT:   IN AL, DX          ;读入状态信息
        TEST AL, 01H       ;测试状态位的状态
        JNZ WATT           ;状态不为 0,则继续读(表示外设正忙,要等待)
        MOV DX, 03FBH      ;状态为 0,则可输出数据,将输出端口地址送 DX
        MOV AL, [SI]       ;取一个字节数
        OUT DX, AL         ;输出一个字节数据
        INC SI             ;修改地址指针
        LOOP AGAIN         ;若未传送完,则重复执行以上过程
        HIT               ;传送完毕,则结束

```

以上是单个外设利用查询方式进行数据传送的工作过程。但事实上,一个微型计算机系统往往要连接多个外设。这种情况下 CPU 会对外设逐个进行查询,发现哪个外设准备就绪,就对该外设进行相应的数据传送。然后再查询下一外设,依次循环。此时的工作流程如图 7.8 所示。

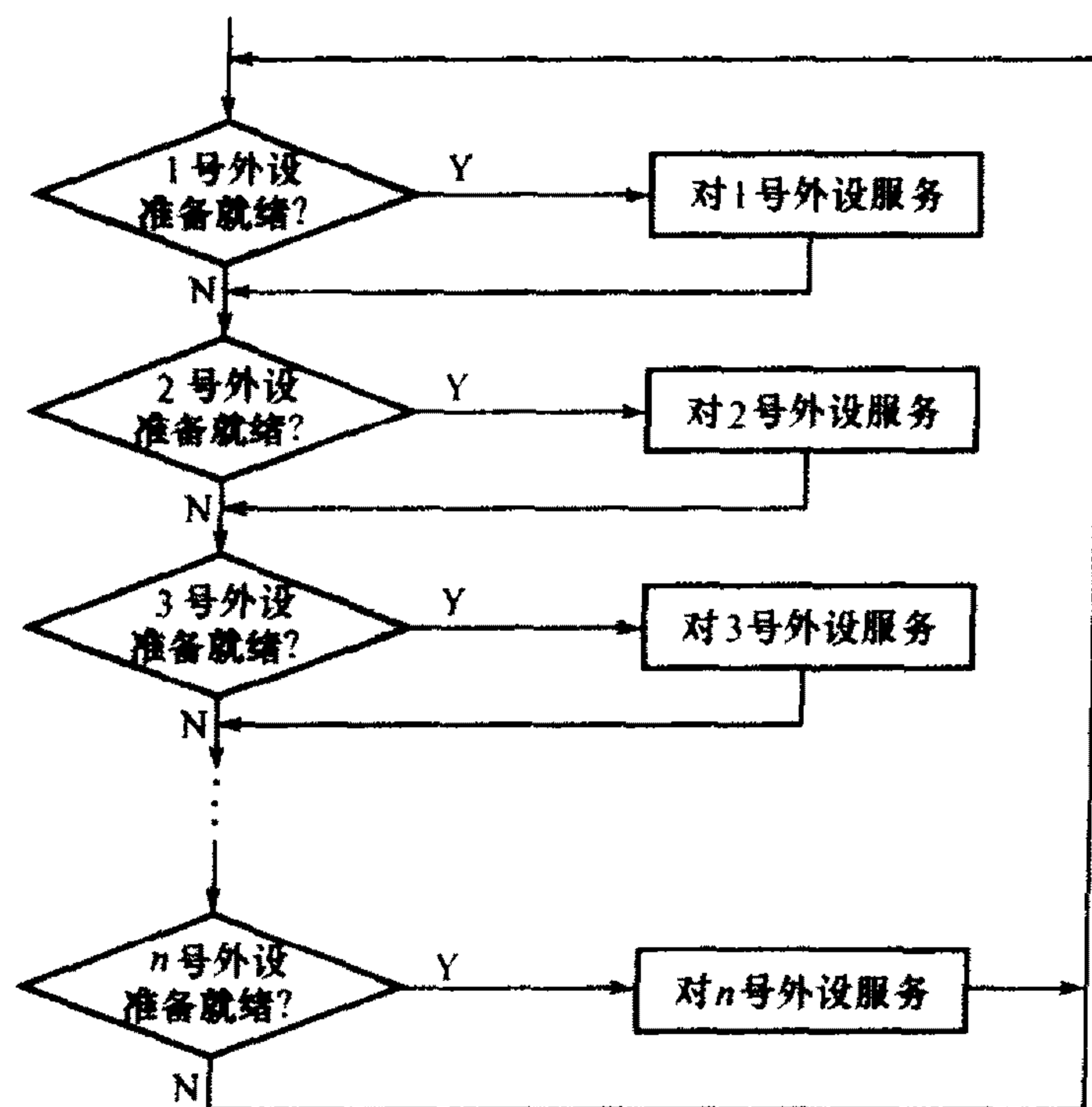


图 7.8 多个外设时的查询方式工作流程

由上述可知,利用查询方式进行数据的输入/输出时,在整个查询过程中 CPU 都不能再做别的事,这就大大降低了 CPU 的效率。而且,假如某一外设刚好在查询过之后就处于就绪状态,那么也必须等到 CPU 查询完所有外设,再次查询此外设时,CPU 才能发现它处于就绪状态,然后才能对此外设服务。这样使得数据交换的实时性较差,对许多实时性要求较高的外设来说,就有可能丢失数据。

查询工作方式的主要优点是能够保证主机与外部设备在工作上的同步,主要缺点是浪费 CPU 的时间,实时性较差。故多用于简单、慢速的或对实时性要求不高的外部设备。

7.2.2 中断控制方式

所谓中断是由于系统中某些突发的或异常的情况发生,强迫 CPU 暂停(或中断)正在执行的程序而转去进行相应的服务,并在服务结束后回到原来被中断的地方继续执行原来的程序。这样一个过程称为中断响应和处理过程。

程序控制的输入/输出方式是由 CPU 作为主动的一方去管理外部设备,在管理的过程中 CPU 不做别的事情。其中,无条件的传送方式适用于简单、慢速的外部设备,其软、硬件都比较简单,但适用范围较窄,而且在 CPU 与外设不同步时容易出错;查询工作方式则将大量的时间耗费在读取外设状态及进行检测上,真正用于传送数据的时间很少,从而降低了 CPU 的效率,并在多个外设的情况下无法对一些外部事件进行实时响应。因此,它也多用于慢速和中速外设。这种输入/输出方式对具有多外设且要求实时性较强的计算机控制系统是不适合的。

正因为此,才引进了中断的概念。即 CPU 并不主动介入外设的数据传输工作,而是由外部设备在需要进行数据传送时向 CPU 发出中断请求,CPU 在接到请求后若条件允许,则暂停(或中断)正在进行的工作而转去对该外设服务,并在服务结束后回到原来被中断的地方继续原来的工作。这种方式能使 CPU 在没有外设请求时进行原有的工作,有请求时才去处理数据的输入/输出,从而提高了 CPU 的利用率。但有一点要注意,就是在 CPU 对外设服务结束后要能够回到原来被中断的地方,这就要求在响应中断前必须将返回地址(即中断时 CPU 将要执行的指令的地址)和程序运行状态保存起来,以保证正确返回。这个过程称为断点保护。

利用中断方式进行数据传送,不仅大大提高 CPU 效率,还能够对外设的请求做出实时响应。尤其是在外设出现故障、不立即进行处理就有可能造成严重后果的情况下,利用中断方式,可以及时做出处理,避免了不必要的损失。有关中断的概念、工作原理及中断源分类等将在本章的第 3 节做详细的讨论。

7.2.3 直接存储器存取方式(DMA)

虽然采用中断方式能大大提高 CPU 的利用率,但与程序控制方式一样,实际的数据传送过程还是需要 CPU 执行程序来实现,即 CPU 通过输入/输出指令将数据从内存(或外设)读到累加器,再写入到接口(或内存)中。因此,从这个角度讲,中断控制方式也可认为是程序控制输入/输出方式的一种。另外,采用中断方式每进行一次数据传送,都需要保护断点、保护现场等。若再考虑到修改内存地址,判断数据块是否传送完等因素,CPU 传送一个字节速度就不会很高。这对于一些高速外设及批量数据交换(如磁盘与内存的数据交换)来说是不能满足要求的。

对需要高速数据传送的设备,希望外设能够不通过 CPU 而直接与存储器进行信息交换,这就是直接存储器存取(Direct Memory Access, DMA)方式。DMA 方式的基本思想是在外设与主存储器之间开辟直接的数据传送通路。在正常工作时,由 CPU 控制总线,所有的工作周期都用于执行 CPU 程序。当外部设备有数据准备好需要与主存进行交换时,则向 CPU 申请占用一个总线周期。在这个周期之后,CPU 又继续控制总线,执行原来的程序,直到外设有下一个准备好的数据。如此反复,一直到整个数据块传送完毕。

实现 DMA 传送工作是通过特殊的硬件电路——DMA 控制器来控制完成的,由它给出每次进行数据传送所针对的主存地址,并统计已传送数据的个数,以确定数据是否传送结束。在传送开始和结束时通过中断方式进行处理,而在数据传送的过程中则不需要 CPU 干涉,完全由 DMA 控制器(DMAC)控制,使主存被并行工作的 CPU 和输入/输出子系统共享。下面简要介绍 Intel 公司生产的典型的 DMA 控制器 8237 的功能及工作过程。

1. DMA 控制器的功能

通常情况下,系统的地址总线、数据总线和一些控制信号,如 $\overline{M}/\overline{IO}$ 、 \overline{MR} 、 \overline{RD} 等是由 CPU 管理的,而在 DMA 方式下,就要求 DMA 控制器接管这些信号线的控制权,这就要求 DMA 控制器具有以下功能:

- ① 收到接口发出的 DMA 请求后,DMA 控制器要向 CPU 发出总线请求信号 HOLD(高电平有效),请求 CPU 放弃总线的控制;
- ② 当 CPU 响应请求并发出响应信号 HLDA(高电平有效)后,这时 DMA 控制器要接管总线的控制权,实现对总线的控制;
- ③ 能向地址总线发出内存地址信息,找到相应单元,并能够自动修改其地址计数器;
- ④ 能向存储器或外设发出读/写命令;
- ⑤ 能决定传送的字节数,并判断 DMA 传送是否结束;
- ⑥ 在 DMA 过程结束后,能向 CPU 发出 DMA 结束信号,将总线控制权交还给 CPU。

2. DMA 控制器的工作过程

DMA 的工作过程大致如下:

① 当外设准备好,可以进行 DMA 传送时,外设向 DMA 控制器发出 DMA 传送请求信号 (DRQ);

② DMA 控制器收到请求后,向 CPU 发出“总线请求”信号 HOLD,表示希望占用总线;

③ CPU 在完成当前总线周期后会立即对 HOLD 信号进行响应。响应包括两个方面:一是 CPU 将数据总线、地址总线和相应的控制信号线均置为高阻态,由此放弃对总线的控制权,另一方面,CPU 向 DMA 控制器发出“总线响应”信号 (HLDA);

④ DMA 控制器收到 HLDA 信号后,就开始控制总线,并向外设发出 DMA 响应信号 DACK;

⑤ DMA 控制器送出地址信号和相应的控制信号,实现外设与内存或内存与内存之间的直接数据传送(例如,在地址总线上发出存储器的地址,向存储器发出写信号 $\overline{\text{MEMW}}$,同时向外设发出 I/O 地址、 $\overline{\text{IOR}}$ 和 AEN 信号,即可从外设向内存传送一个字节);

⑥ DMA 控制器自动修改地址和字节计数器,并据此判断是否需要重复传送操作。规定的数据传送完后,DMA 控制器就撤销发往 CPU 的 HOLD 信号。CPU 检测到 HOLD 失效后,紧接着撤销 HLDA 信号,并在下一时钟周期重新开始控制总线,继续执行原来的程序。

图 7.9 所示的是 DMA 方式中存储器写的总线周期时序,图中 DMA 控制器在 HLDA 有效期间获得总线控制权。在 S_3 周期和 S_4 周期之间插入了一个等待的时钟周期 S_w 。在 $S_1 \sim S_3$ 期间,DMAC 送出地址信号和控制信号,选中写入的内存地址单元,将外设提供的有效

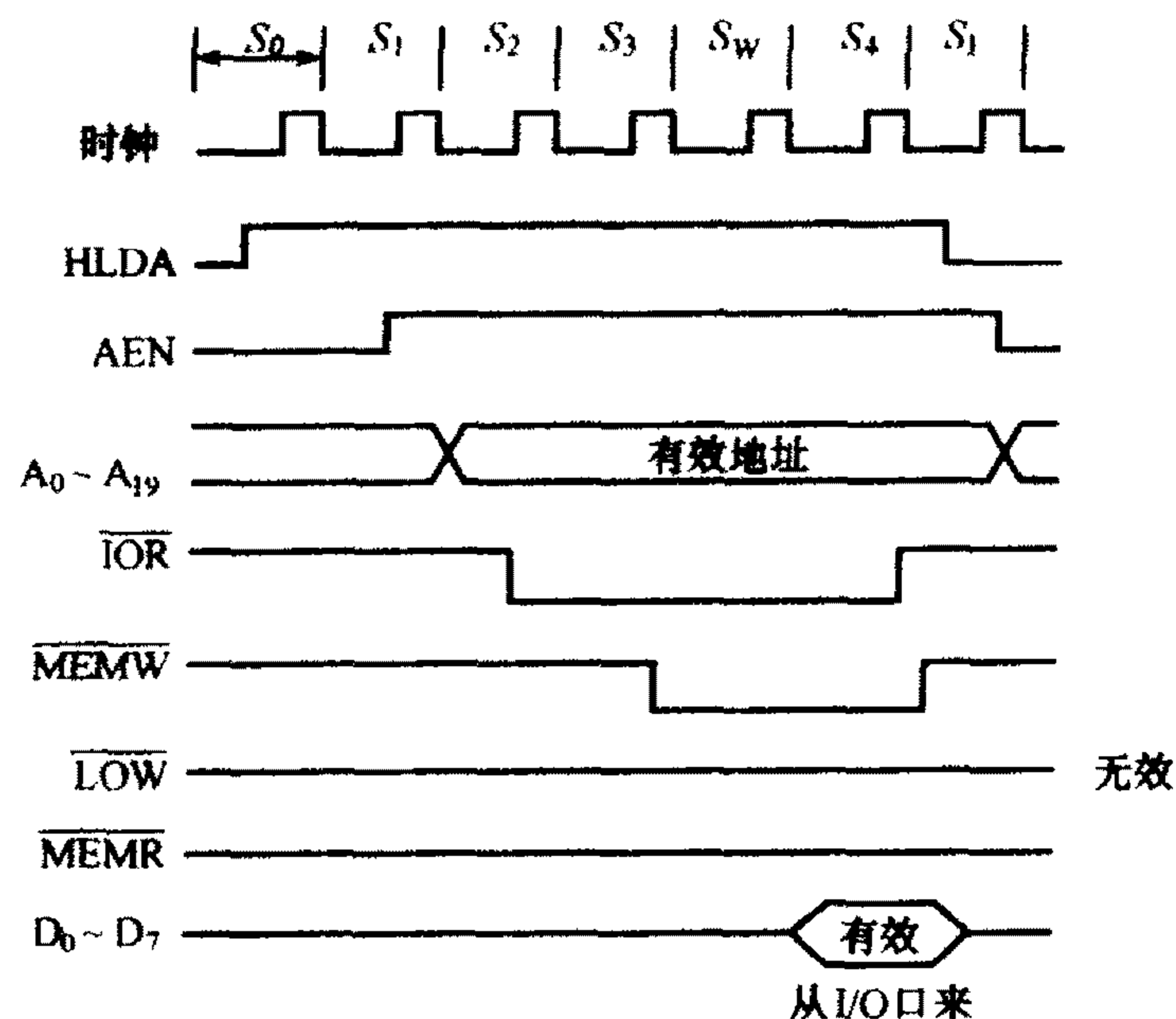


图 7.9 DMA 存储器写的总线周期时序

数据写入规定的内存单元。

为了进一步说明 DMA 的传送过程,图 7.10 给出了一个 DMA 存储器写操作的简要原理图。这里要注意两点,一是 DMA 传送前,CPU 必须告诉 DMA 控制器传送是在哪两个部件之间进行,传送的内存首地址以及传送的字节数是多少;二是在 DMA 传送时,DMA 控制器只负责送出地址及控制信号,而数据传送是直接在接口和内存间进行的,并不经过 DMA 控制器。对于内存与内存间的 DMA 传送,是先用一个 DMA 的存储器读周期将数据由内存读出,放在 DMA 控制器的内部数据暂存器中,再利用一个 DMA 的存储器写周期将该数据写到内存的另一区域。

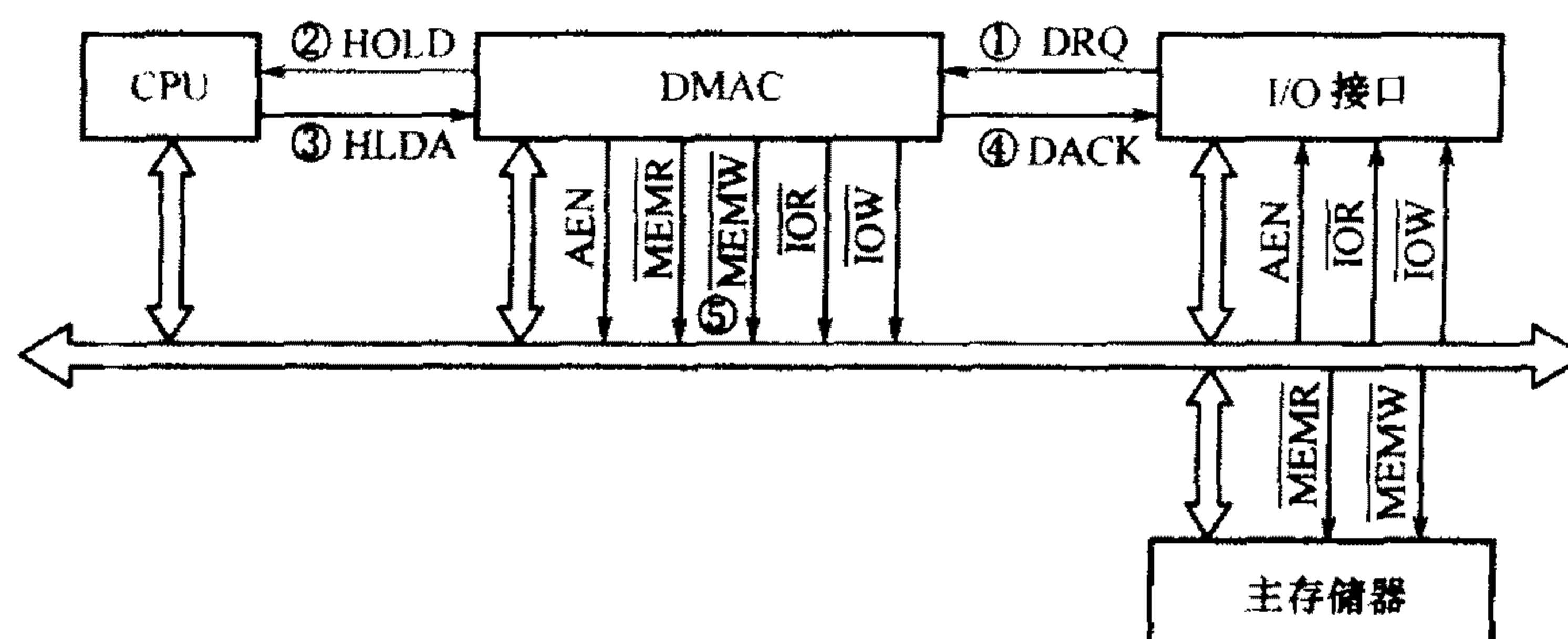


图 7.10 DMA 传送原理示意图

虽然 DMA 方式实现了外部设备与主存之间的直接数据传送,但它也存在着一定的局限性。首先是对外部设备的管理和对某些操作的控制仍需 CPU 完成。另外,随着系统功能的不断扩展,外部设备的种类和数量都在不断增加,对它们的管理和控制也变得越来越复杂,亦即对 DMAC 的要求越来越高,有时甚至需要多个 DMAC 同时使用,从而引起对主存访问冲突。因此,为了解决这些问题,在大型计算机系统中通常都设置了专门用于和外设进行数据通信的硬件装置,即 I/O 通道。

7.2.4 I/O 通道控制方式

“通道”是一个专用名词,它不仅是指具有专门的指令系统、能独立进行操作并控制完成整个输入/输出过程的硬件装置,而且还是一种概念,一种具有综合性及通用性的输入/输出方式,它也代表了现代计算机组织向功能分布方向发展的初始发展阶段。

由于 I/O 通道具有自己独立的指令系统,且可独立完成指令规定的操作,因此它也可以被看做一个简单的处理机。但它的指令系统是很简单的,一般只有几条与输入/输出操作有关的命令,而且它的启动、停止及工作状态的转换等还是要在 CPU 的统一指挥下进行,某些

操作如错误校验、码制转换等仍要由 CPU 完成,所以它还不是一个完全独立的处理机,只能说是从属于 CPU 的一个专用 I/O 处理机。图 7.11 所示的是通道控制方式下的系统结构,在目前的大型计算机系统中,“通道”已是一个完全独立的输入/输出处理机,或者说就是一个小型的通用计算机,它可基本独立于主机工作,完成输入/输出控制及通道不能独立完成的码制转换、错误校验、格式处理等。

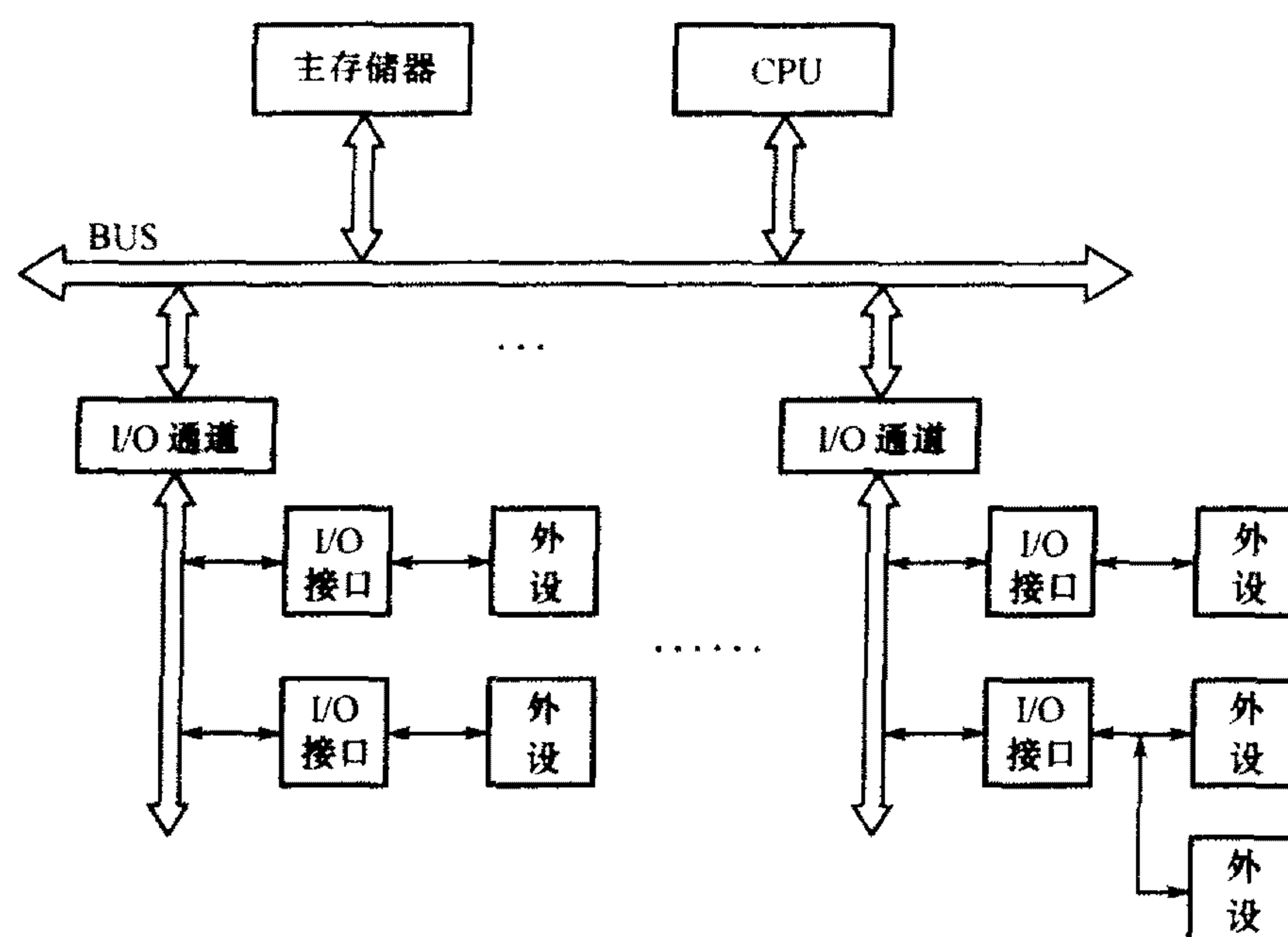


图 7.11 通道控制方式下的系统结构

7.3 中断技术

中断技术在计算机中应用极为广泛,它不仅可用于数据传输,以提高数据传输过程中 CPU 的利用率,还可以用来处理一些需要实时响应的事件,例如异常事件、时钟、掉电、特殊状态等。在操作系统中,还使用中断来进行一些系统级的特殊操作,如虚存中页面的调入和调出等。

7.3.1 中断的一般概念

1. 中断及中断源

中断是指在 CPU 执行程序的过程中,由于出现某种异常情况或随机事件(包括 CPU 内部的和 CPU 外部的事件),使得 CPU 暂时停止正在执行的程序,而转去执行一个用于处理该

事件的程序——称为中断服务程序(或中断处理程序),并在处理完后又返回被中止的原程序断点处继续执行,这一过程就称为中断。

引起中断的事件,即发出中断请求的来源,称为中断源。中断源的种类概括起来可以分为如图 7.12 所示的两大类:内部中断和外部中断。

内部中断源在处理器内部,根据其产生中断的原因又可分为异常中断和软中断,主要包括这样几个方面:

- ① CPU 执行指令时产生的故障异常 如除数过小等;
- ② 陷阱中断 如 TF 标志位等于 1 产生的单步中断、溢出

等;

- ③ 特殊操作引起的异常 如存储器越界、缺页等;
- ④ 由程序员安排在程序中的软件中断指令 INT n 所引起中断。

外部中断源则主要包括:

- ① 输入/输出设备 如键盘、打印机、鼠标等;
- ② 数据通道 如磁盘、数据采集装置、网络等;
- ③ 实时时钟 如定时/计数器时间到;
- ④ 故障源 如掉电、硬件错、存储器奇偶校验错等。

对内部中断来说,中断的控制完全是由 CPU 内部处理来实现的;而对于外部中断,则是利用 CPU 的两条中断输入信号线 INTR 和 NMI 来告诉 CPU 已发生了中断事件。

INTR 称为可屏蔽中断请求输入信号,即 CPU 能够根据中断允许标志位 IF 的状态来决定是否响应该请求信号。如果 $IF = 1$,CPU 就在当前指令执行完后响应该中断请求;若 $IF = 0$,则 CPU 就不予响应,亦即该中断请求被屏蔽。所以,也将 $IF = 1$ 称为开中断, $IF = 0$ 称为关中断。

NMI 称为非屏蔽中断请求输入信号,上升沿有效。所谓“非屏蔽”就是必须响应,所以它不受标志位 IF 的约束。只要 CPU 在正常地执行程序,它就一定会在当前指令执行完后响应 NMI 的请求。

事实上,“中断”在日常生活中也是很常见的。例如,当你正在看书时,门铃和电话铃同时响了,这时你必须对这两个事件做出反应,并迅速做出判断:是先接电话还是先开门。假如你认为开门比较紧急,就会暂时停止看书(你可能还会在正看的页码处夹上书签),而先去开门,然后去接听电话,这两个事件处理完后,再从原来中断的地方接着看你的书。

2. 中断的作用

在微型计算机系统中,中断是一种特殊的程序执行过程,是处理器与外部设备进行信息交换的一种方式。它主要起着以下几个方面的作用:

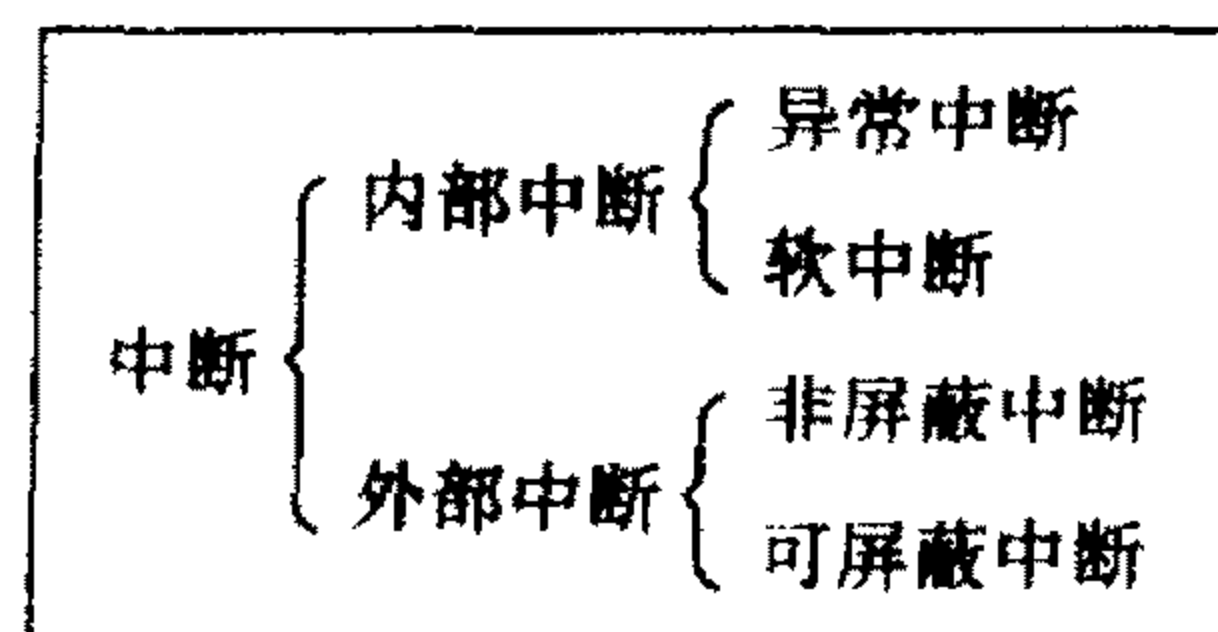


图 7.12 中断源的种类

1) 使 CPU 与 I/O 设备实现并行工作

由于中断是在外设需要时向 CPU 提出申请, CPU 再转去为它服务。换句话说,就是在没有中断申请时 CPU 可以执行自己的程序。外部设备的速度相对处理器来讲,一般都要慢很多。在它慢慢“消化”CPU 传送给它的信息时, CPU 可以并行工作,直到外设再次提出服务请求。

例如,打印机在完成一行打印后,向 CPU 发出中断请求,如果 CPU 响应请求,则会暂停正在执行的程序而转入打印服务程序,即将下一行要打印的字符传送到打印机控制器,并启动打印机工作。由于 CPU 的速度很高,这个过程是很短的,一般都在微秒级;但打印机打印一行字符需几毫秒到几十毫秒的时间,在此期间, CPU 就可以运行自己的程序。所以,从宏观上看,采用中断控制方式可使得 CPU 与 I/O 设备并行工作。CPU 与打印机并行工作的时序如图 7.13 所示。

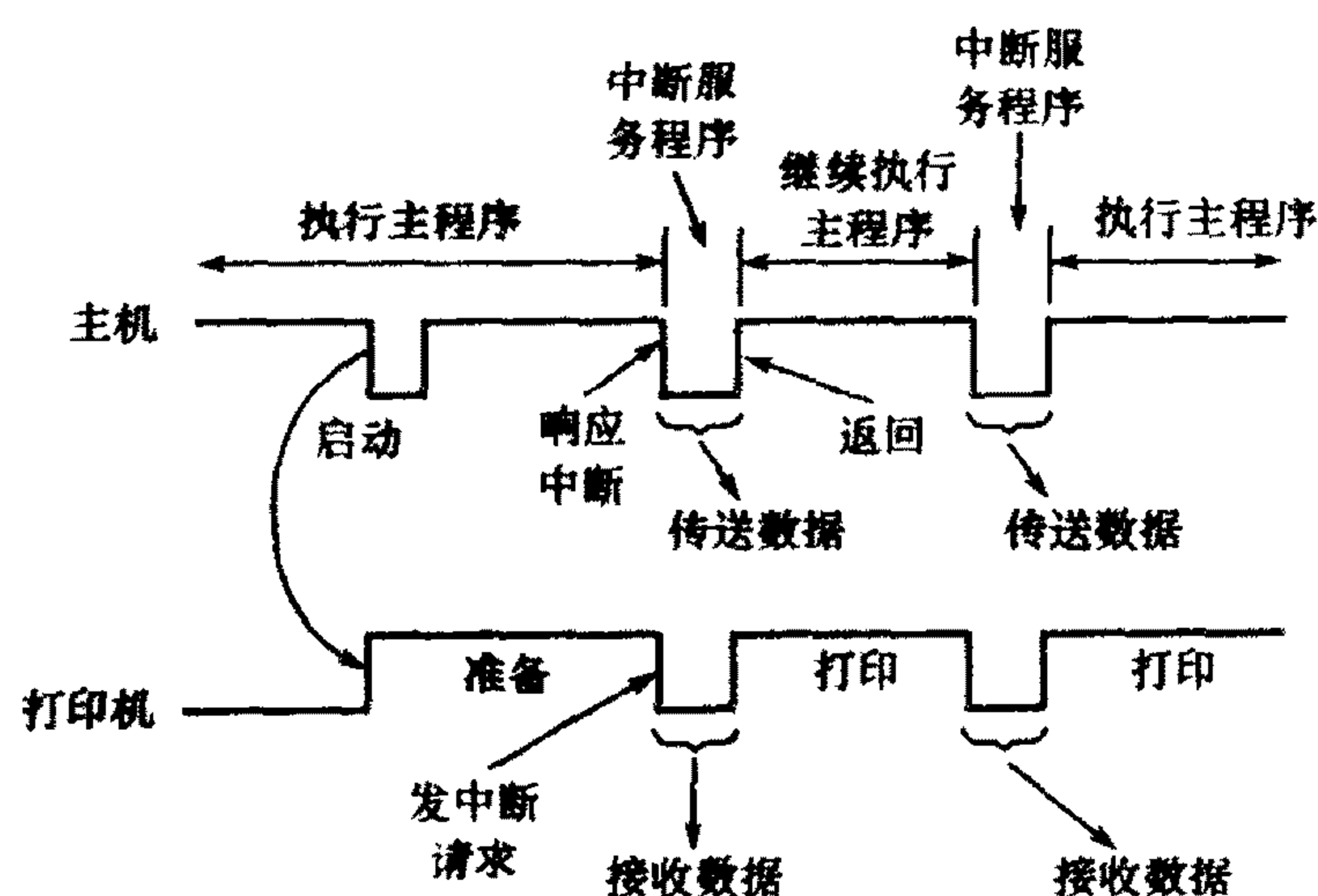


图 7.13 CPU 与打印机并行工作时序图

2) 自动故障处理

若计算机在运行中出现硬件故障,中断系统会立刻发出中断请求, CPU 在响应中断后转去执行相应的故障处理程序。

3) 实现实时处理

所谓实时处理,是指尽可能及时地去处理某个突然发生的事件。实时处理大量用于计算机过程控制中。例如,当生产线上的温度或压力发生变化时,要求能及时地输入计算机进行处理,并发出相应的控制信号。而利用中断技术就能够实现这一点。

4) 实现人机通信

利用中断系统可以方便地、有效地实现人机通信。例如,在计算机运行过程中,可通过

中断获取中间运算结果、检测机器工作状态等。

5) 实现多道程序和分时操作

多道程序的切换运行需要借助中断系统。在一道程序运行时,可通过中断系统切换到另一道程序运行。

除以上几点外,还可利用软中断指令实现对操作系统功能模块的调用。在多处理机系统中,处理机与处理机之间的信息交流和任务切换也是通过中断来实现的。

7.3.2 中断响应的工作过程

计算机的中断响应过程包括中断请求、中断源识别、中断响应和中断处理这几个步骤。下面以外部可屏蔽中断为例进行讨论。

1. 中断请求

当发生异常情况或某些随机事件使外部设备需要 CPU 提供服务时,首先向 CPU 的中断请求输入端发送一个有效的中断请求信号。中断请求信号分为边沿触发和电平触发。边沿触发指的是 CPU 根据中断请求端上有无从低到高或从高到低的跳变来决定中断请求信号是否有效;电平触发指的是 CPU 根据中断请求端上有无稳定的电平信号(高电平还是低电平取决于 CPU 的设计)来确定中断请求信号是否有效。一般来说,CPU 能够即时响应的中断可以采用边沿触发,而不能即时响应的中断则应采用电平触发,否则中断请求信号就会丢失。8086CPU 的 NMI 为边沿触发,而 INTR 为电平触发。为了保证产生的中断能被 CPU 处理,INTR 中断请求信号应保持到该请求被 CPU 响应为止。CPU 响应后,INTR 信号还应及时撤除,以免造成多次响应。

2. 中断源识别

一般地讲,一个计算机系统包含有多个中断源。由于中断产生的随机性,使得有可能在某一时刻有两个以上的中断源同时发出中断请求,但 CPU 往往只有一条中断请求线,并且任一时刻只能响应并处理一个中断。到底先响应哪一个中断源的请求?在现代计算机系统中是采取为每个中断源设置一个优先级别的方法,再根据其级别的高低确定哪个先被响应。因此,中断源的识别需要解决这样两个问题:

① 中断源确定 在只有一个中断源提出中断请求时,需要找出是哪一个中断源发出了请求;

② 优先级判定 在有多多个中断源提出中断请求时,要确定提出请求的中断源的优先级顺序,以保证先响应高优先级的中断,处理完后再依次响应低优先级中断。这就是所谓的中断判优。

中断优先级判定的方法一般有两种:软件判优和硬件判优。

1) 软件判优

所谓软件判优是指通过软件来安排各中断源的优先级别。软件判优需要相应电路的支持,如图 7.14 所示。在电路中,各中断源的中断请求信号 IRQ 先锁存到中断请求寄存器中,再通过或门后送到 CPU 的 INTR 中断请求输入端。同时,各中断请求信号的状态经并行 I/O 端口输入 CPU。

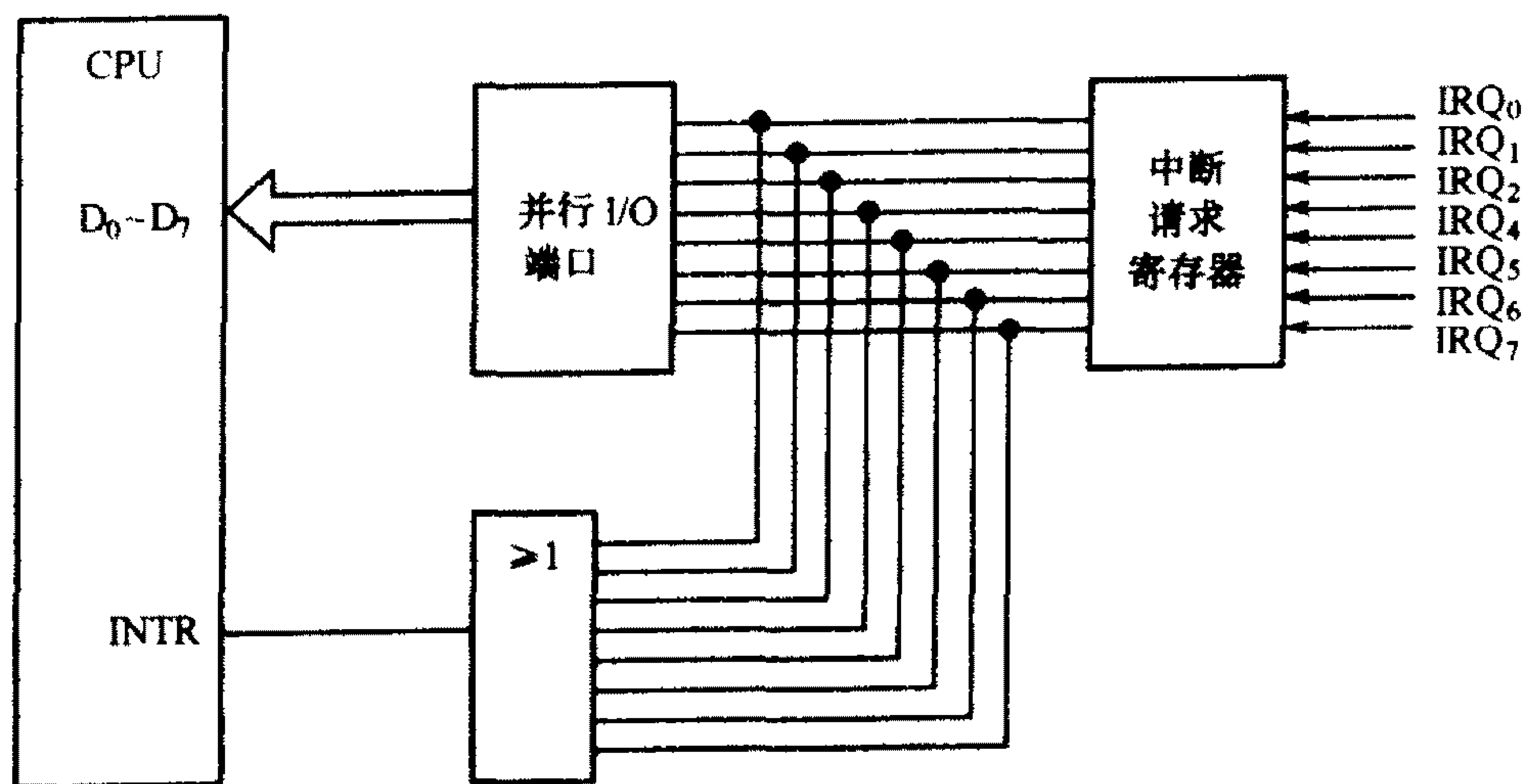


图 7.14 软件判优的电路原理图

发出中断请求的中断源的 IRQ 状态为“1”（高电平），状态“1”经或门送到 CPU 的 INTR 引出线上，CPU 响应中断后，首先用软件将各中断源通过并行端口传送的请求信号状态读入，再逐位查询各 IRQ 的状态是“1”还是“0”，为“1”，则表示它对应的中断源提出了中断请求，下一步就转入该中断源的中断服务子程序。在这里，查询的次序就反映了各中断源优先级别的高低，先被查询的中断源优先级别最高，后被查询的优先级依次降低。其工作流程如图 7.15 所示。

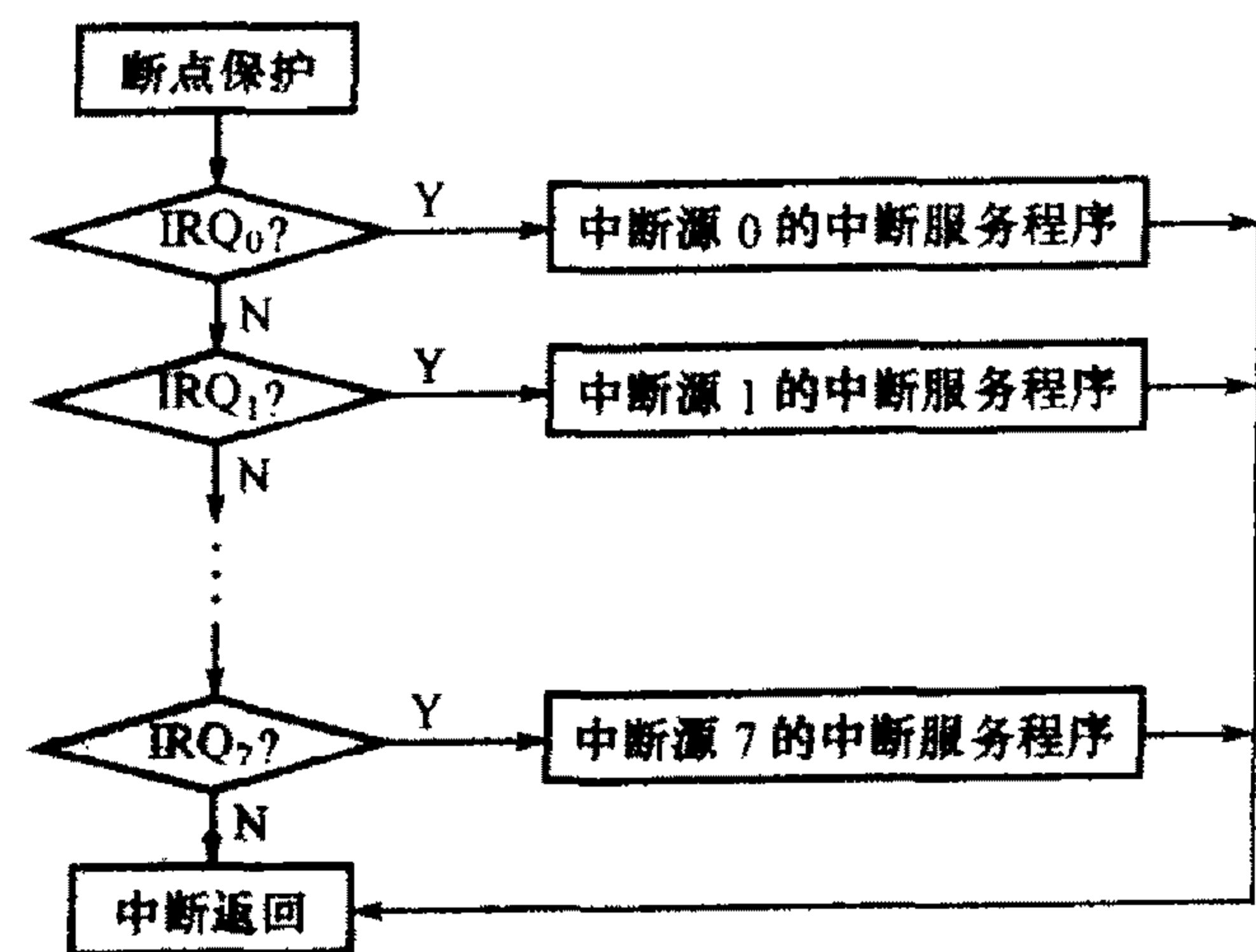


图 7.15 软件中断优先级查询流程图

这种判优方法的硬件电路简单，优先级安排灵活，但软件判优所花时间较长，在中断源较多的情况下会影响到中断响应的实时性。硬件判优则可较好地克服这个缺点。

2) 硬件判优

硬件判优是指利用专门的硬件电路或中断控制器来安排各中断源的优先级别。硬件优

先级判优电路形式很多。这里介绍两种常用的硬件判优方法。

(1) 中断向量法

这种方法是为每一个中断源分配的一个编号,称为中断向量码或中断类型码,系统通过不同中断源提供的不同的中断向量码来识别提出请求的中断源,并通过该向量码确定与中断源相对应的中断服务子程序的入口地址。

电路中用一个中断优先级判别器来判别哪个中断请求的优先级最高,然后在 CPU 响应中断时把此中断源所对应的中断向量码送给 CPU, CPU 根据中断向量码找到相应的中断服务程序入口,执行相应的处理程序。

与 8086CPU 配套的 8259 芯片就是一种可编程的中断控制器, CPU 通过向它写入控制字,可实现对多达 64 级的中断进行优先级管理。

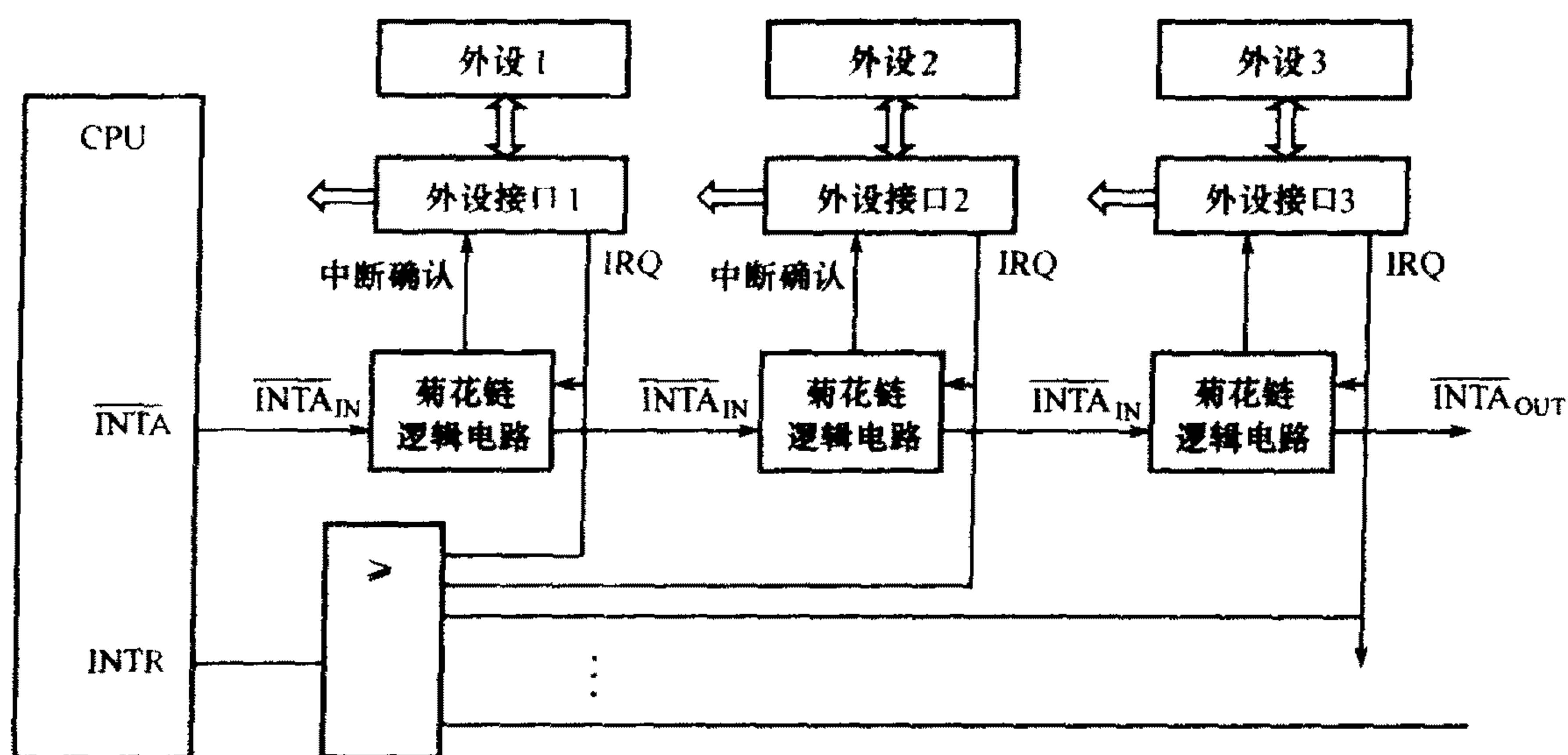
(2) 链式判优电路

链式判优的基本思想是将所有的中断源构成一个链(称为菊花链),各中断源在链中的前后顺序是根据中断源优先级别的高低来排列的,排在链前面的高优先级别的中断会自动封锁后边低优先级别的中断。链式优先级排队电路如图 7.16(a)所示。电路中,每个外设对应的接口都有一个中断逻辑电路, CPU 响应中断时发出的 \overline{INTA} 信号沿着这些逻辑电路串接成的菊花链从前往后传递。

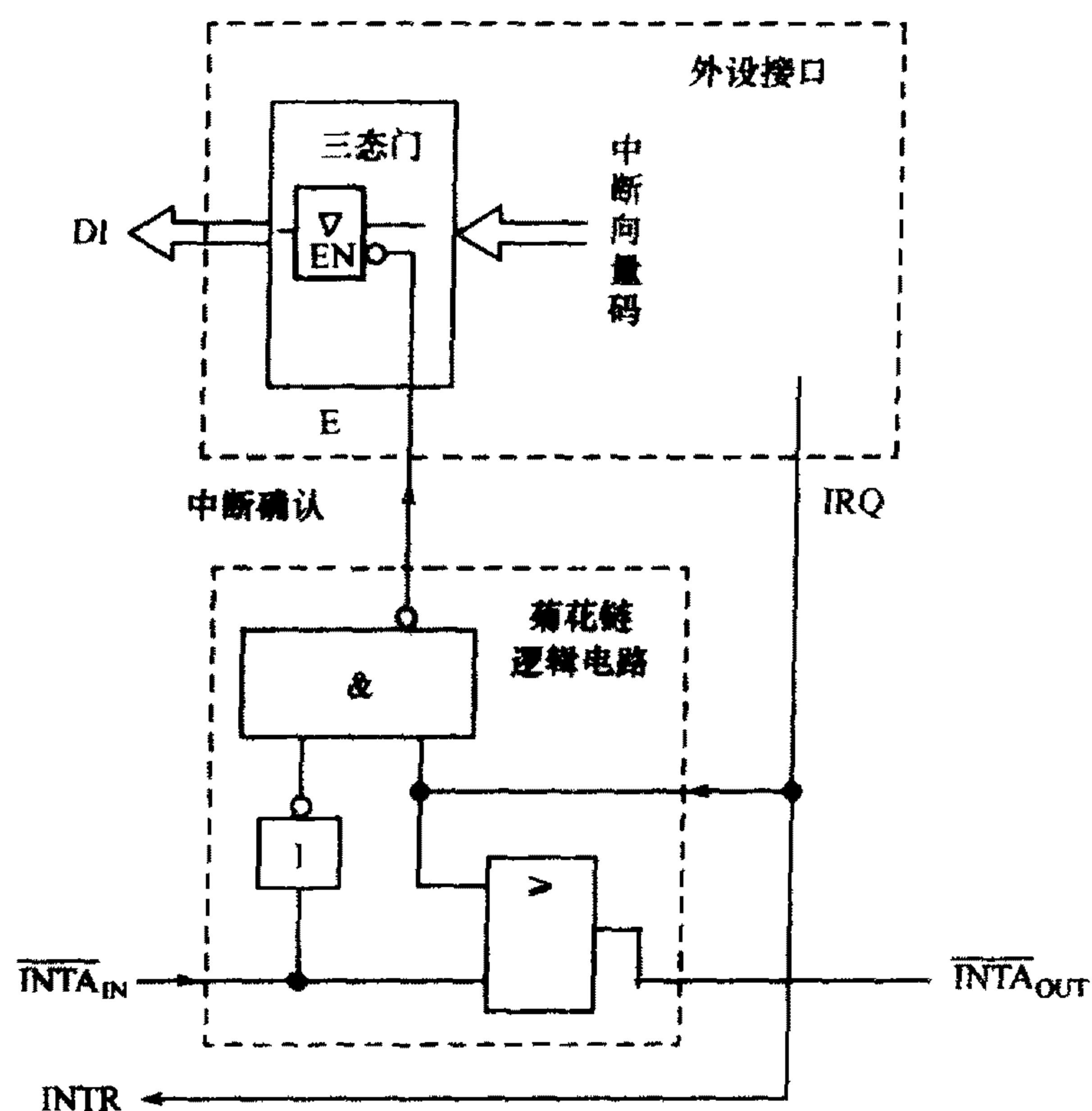
从图 7.16(b)中可以看出,当某个外设有中断请求时, CPU 如果允许中断,则会发出低电平有效的 \overline{INTA} 信号,此时若菊花链前端的外设没有发出中断请求信号,即对应的 IRQ 为低电平,则菊花链逻辑电路(图 7.16(b))中的与非门输出为“1”,外设接口中传送中断向量码的三态门处于关断状态。这样,中断响应信号 \overline{INTA} 就会原封不动地向后传递,一直传到发出中断请求的外设。当然,该外设的 IRQ 端这时为“1”,使外设接口中的三态门导通,将该中断源的中断向量码送上数据总线。在此同时,这个外设发出的状态为高电平的 IRQ 中断请求信号会使 \overline{INTA}_{OUT} (即后一级菊花链的 \overline{INTA}_{IN}) 为高电平,自动封锁了后面设备的中断逻辑电路,使 \overline{INTA} 信号不再传到后面的外设(其后外设的 \overline{INTA} 输入端全部为“1”信号)。

菊花链电路中各个外设的中断优先级就由其链中的位置决定,处于菊花链前端的比处于链条后端的优先级高。菊花链前端发出中断请求的外设截获 \overline{INTA} 信号后,将自己的中断类型码放到数据总线上供 CPU 读取, CPU 根据此类型码计算出中断服务程序的入口地址,然后转去执行相应的处理程序。

当多个外设同时发出中断请求信号时,根据电路分析可知,菊花链中位置靠前的外设将截获 \overline{INTA} 信号,而排在菊花链中较后位置的外设就收不到 \overline{INTA} 信号,因而暂时不会被处理。若 CPU 正执行某个中断服务程序时,又有级别较高的外设提出中断请求,由于菊花链电路中级别低的外设不能封锁级别高的外设得到中断响应信号,故仍可响应该中断请求,所以,此电路也能实现中断嵌套。



(a) 链式判优电路原理图



(b) 菊花链逻辑电路

图 7.16 菊花链中断判优电路

3) 中断嵌套

中断嵌套类似于子程序嵌套,即高优先级别的中断可以中断低优先级别的中断,出现一层套一层的现象。大部分中断控制电路在解决中断优先级的同时也实现了中断嵌套。中断

嵌套的层数一般不受限制,但设计中断程序时要注意留有足够的堆栈空间,因为每一层嵌套都要用堆栈来保护断点及相应的参数,使得堆栈内容不断增加,若堆栈空间过小,中断嵌套层次较多时就会产生堆栈溢出现象,使程序运行失败。

3. 中断响应

中断优先级确定后,发出中断的中断源中优先级别最高的中断请求就被送到 CPU 的中断请求引出线上,下一步就是 CPU 对进入的中断请求做出响应。CPU 在每条指令执行的最后一个时钟周期检测中断请求引出线上有无中断请求信号,但并不是在任何时刻、任何情况下 CPU 都能对中断请求进行响应。能否响应中断请求,必须满足以下 4 个条件:

① 一条指令执行结束 CPU 在每条指令执行的最后一个时钟周期对中断请求进行检测,当满足本条件和下述 3 个条件时,执行完该指令,CPU 就立即响应中断。

② CPU 处于开中断状态 只有当控制标志位 $IF = 1$,即处于开中断状态时,CPU 才有可能响应可屏蔽中断(INTR)请求(对从 NMI 端输入的中断请求无此约束)。

③ 当前没有发生复位(RESET)、保持(HOLD)和非屏蔽中断请求(NMI) 在复位或保持状态时,CPU 不工作,不可能响应中断请求;而 NMI 的优先级比 INTR 高,当两者同时产生时,CPU 会响应 NMI 而不响应 INTR。

④ 若当前执行的指令是开中断指令(STI)和中断返回指令(IRET),则执行完它们之后要再执行一条指令,CPU 才能响应 INTR 请求。另外,对前缀指令,如 LOCK、REP 等,CPU 会把它们和它们后面的指令看做一个整体,直到这个整体指令执行完,方可响应 INTR 请求。

中断响应时,CPU 除了要向中断源发出中断响应信号外,还要做下述两项工作:

- 断点保护,包括 CS、IP 和 PSW(FLAGS),即将它们的内容压入堆栈。这主要是保证中断结束后能返回被中断的程序。断点保护的顺序是: FLAGS—CS—IP;
- 获得中断服务程序的入口地址。

4. 中断处理

中断处理是由硬件系统和中断服务子程序来完成的。中断服务子程序在形式上与一般的子程序差不多,不同之处在于:

- ① 中断服务子程序只能是远过程(类型为 FAR);
- ② 中断服务子程序要用 IRET 指令返回被中断的程序。

中断处理一般包括以下几个步骤:

① 关中断 即进入不能再次响应中断的状态。因为在此之后要进行断点保护和现场保护,而在“保护”的过程中,即使有更高级的中断源提出中断请求,CPU 也不能再响应。否则容易使断点及现场的保护不完整,如果这样,则在中断服务程序执行完后,就无法正确恢复现场并继续执行原程序。

② 保护断点和现场 为了在中断服务程序结束后能正确返回原被中断处,需要在进入中断服务程序之前,将标志寄存器的内容及被中断点的段地址 CS 和偏移地址 IP(断点)压入堆栈保护起来。另外,中断服务程序中要用到的寄存器的原内容也要压入堆栈保存。

断点的保护一般是由硬件来实现的,即硬件系统自动实现将断点地址和 FLAGS 内容压入堆栈。其他要保护的寄存器的内容则是在中断服务程序中由软件实现。

③ 获取中断向量码 通过读入中断向量码确定提出请求的中断源,并进一步得到中断服务程序的入口地址。

④ 开中断 若紧接着要执行中断服务程序允许中断嵌套,则需用指令开中断(如 8086 中的 STI 指令),使 $IF=1$,使得在执行中断服务程序的过程中能够响应更高级的中断请求。

⑤ 执行中断服务程序 不同的中断,其中断处理程序也各不相同,编程人员可根据中断处理的需要来编写。但中断服务处理程序不宜过长和过于复杂,在中断处理程序中停留的时间越短越好,否则程序运行时既容易出乱,也影响对其他中断源的及时处理。通常的处理方法是,在中断服务程序中只执行那些必须执行的操作,而其他相关操作可放到中断服务程序外去执行(例如放到主程序中)。

⑥ 关中断 相应的中断处理指令执行结束后需要关中断,以确保有效地恢复被中断程序的现场。在 8086CPU 中,关中断指令为 CLI。

⑦ 中断返回 中断返回需执行中断返回指令 IRET,就是把先前保护的断点和现场进行恢复,也即把所保存的有关寄存器内容按压栈的相反顺序从堆栈中弹出,使这些寄存器恢复到中断前的状态。同样地,断点的恢复由硬件系统完成,其他寄存器内容的恢复则由软件实现。在恢复完断点和现场后,要开中断,再返回原程序。

中断处理过程的流程如图 7.17 所示。

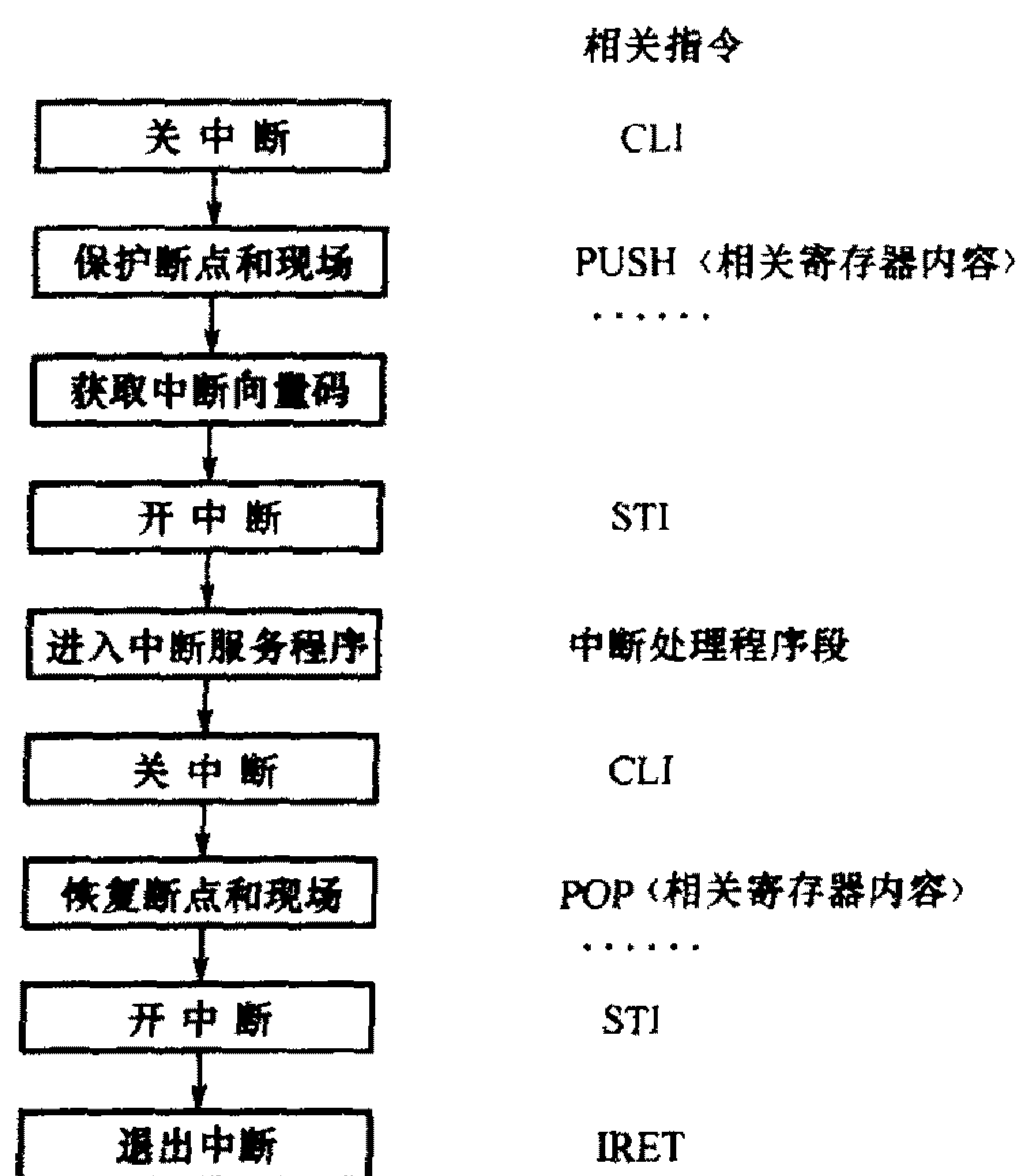


图 7.17 中断处理过程流程图

7.3.3 8086/8088 中断系统

8086/8088 的中断系统功能较强,使用也很灵活,可以处理 256 种不同类型的中断。为了便于识别是哪一个中断源提出请求,系统为每个中断源赋予了一个中断类型码,编号为 0

~ 255。CPU 可根据中断类型码的不同来识别不同的中断源。在前边已经提到,对中断源的识别可以通过软件查询及硬件确定两种方法。软件查询即是在有中断请求时利用程序循环查询每一个中断源,从而确定是哪一中断源提出了请求,它的原理类似于输入/输出方法中的查询工作方式。显然,这种方法在中断源较多的情况下效率比较低。8086/8088 中断系统采用硬件识别中断源的方法,即通过读入中断类型码来确定提出请求的中断源。

1. 中断类型

产生中断请求的中断源可以是在 CPU 内部,也可以在 CPU 外部。由内部中断源产生的中断称为内部中断,包括执行中断指令引起的中断或处理器执行指令时出现错误等引起的异常;相应地,来自 CPU 外部的中断请求就称为外部中断。由于外部中断又分为可屏蔽中断和非屏蔽中断,非屏蔽的中断请求可直接送到 CPU 的 NMI 端,而可屏蔽中断请求则通过中断控制器接到 CPU 的 INTR 端。其系统的示意图如图 7.18 所示。

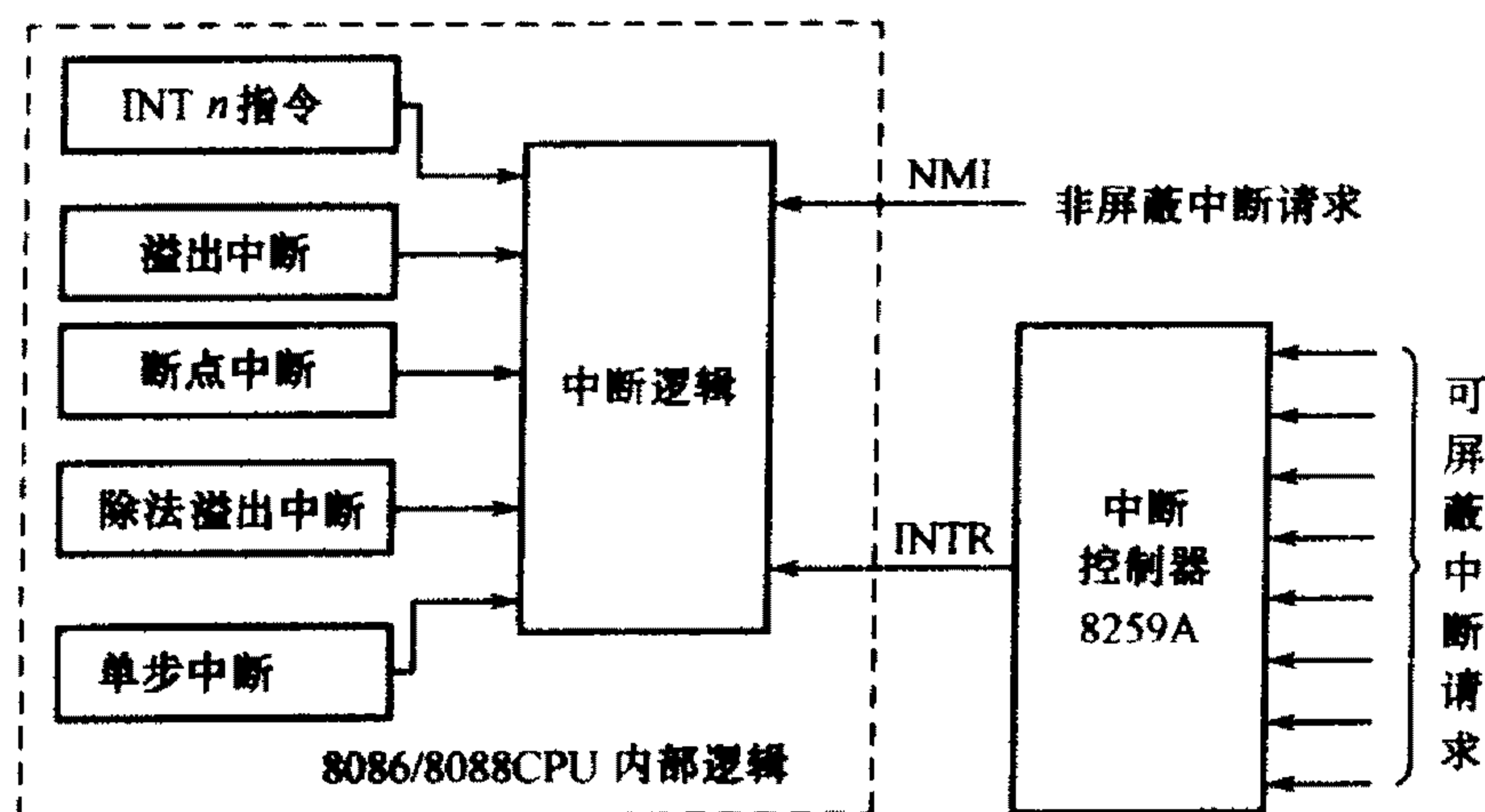


图 7.18 8086/8088 中断源类型

1) 内部中断

内部中断是 CPU 执行了某条指令或者软件对标志寄存器中某个标志位的设置而产生的,由于它与外部硬件电路完全无关,故也称其为软件中断。在 8086/8088 中,内部中断可分为五种类型:

(1) 除法溢出中断——0 型中断

8086/8088 执行除法指令时,若发现除数为 0 或商超过了结果寄存器所能表示的最大范围,则立即产生一个中断类型码为 0 的中断,称为除法出错中断。该中断的服务一般由系统软件进行处理。

(2) 单步中断——1 型中断

8086/8088 的标志寄存器中有一位陷阱标志 TF。CPU 每执行完一条指令都会检查 TF

的状态。若发现 $TF = 1$, 则 CPU 就产生中断类型码为 1 的中断, 转向单步中断的处理程序, 即每执行一条指令就会停下。单步中断广泛地用于程序的调试, 使 CPU 一次执行一条指令, 从而能够逐条指令地观察程序运行情况。在程序排错时, 单步中断是一种很有效的调试手段。

对单步中断要注意两点: 一是所有类型的中断在其处理过程中, CPU 都会自动地把状态标志压入堆栈, 然后清除 TF 和 IF。因此当 CPU 进入单步中断处理程序时, 就不再处于单步工作方式, 而以正常方式工作。只有在单步处理结束时, 从堆栈中弹出原来的标志, 才使 CPU 又回到单步方式。二是 8086/8088 指令系统中没有设置或清除 TF 标志的指令, 但指令系统中的 PUSHF 和 POPF 为程序员提供了置位或复位 TF 的手段。置位和复位 TF 的程序段如下:

```
    ;置位 TF 标志
    PUSHF
    POP AX
    OR AX,0100H      ;TF 置为 1
    PUSH AX
    POPF

    ;复位 TF 标志
    PUSHF
    POP AX
    AND AX,0FEFFH    ;TF 置为 0
    PUSH AX
    POPF
```

(3) 断点中断——3 型中断

8086/8088 指令系统中有一条专门用于设置断点的指令 INT 3, CPU 在执行该指令时会产生一个类型码为 3 的中断。INT 3 指令是一条单字节指令, 能够很方便地插入到程序的任何地方, 专门用于在程序中设置断点, 以进行程序调试, 因此也称它为断点中断, 插入 INT 3 指令的地方是断点。在断点中断服务程序中, 可显示有关的寄存器、存储单元等内容, 以便程序员分析到断点为止程序运行是否正确。

(4) 溢出中断——4 型中断

若算术运算指令的执行结果发生溢出 ($OF = 1$), 则执行 INT 0 指令后立即产生一个中断类型码为 4 的中断。4 型中断为程序员提供了处理运算溢出的手段, INT 0 指令通常和算术运算指令配合使用。

(5) 用户自定义的软件—— n 型中断

CPU 执行所有的中断指令 INT n 都会引起内部中断, 其中断类型码由指令中的 n 指定。

INT n 指令统称为软中断指令。除 INT 3 指令(断点中断)外,其余的 INT n 指令的代码为两字节(第一字节为操作码,第二字节为中断类型码)。

实际上,INT n 软中断可以模拟任何类型的中断,在调试那些非 INT n 中断的中断服务程序时,可以用 INT n 指令来模拟它们发出的中断请求,使原本难于调试的中断程序变得非常简单。

内部中断的类型码都是固定的或包含在软中断指令中。除单步中断外,其内部中断的响应不受 IF 状态标志影响,因此内部中断除单步中断外,其优先级都要高于所有的外部中断。

对于用户自定义的 n 型中断,在响应中断后 CPU 用于中断处理的中断服务程序需要用户自行编制。

2) 外部中断

外部中断也称为硬件中断,它是由硬件设备或外设接口产生的。8086/8088 CPU 为外部设备提供了两条硬件中断信号线 NMI 和 INTR,非屏蔽中断和可屏蔽中断请求信号分别从这两个引出线送入 CPU。

(1) 非屏蔽中断

非屏蔽中断由 NMI 引出线上出现的上升沿触发。它不受中断允许标志 IF 的限制,其中断类型码固定为 2。

CPU 接到非屏蔽中断请求信号后,不管当前正在做什么事,都会在执行完当前指令后立即响应中断请求而进入相应的中断处理。非屏蔽中断通常用来处理系统中出现的重大故障或紧急情况,如系统掉电处理、紧急停机处理等。在 PC 机中,若系统板上的存储器或 I/O 通道上产生了奇偶校验错以及 8087 数学协处理器产生异常都会引起一个 NMI 中断。

(2) 可屏蔽中断

绝大多数外部设备提出的中断请求都是可屏蔽中断,可屏蔽中断的中断请求信号从 CPU 的 INTR 端引入,高电平有效。可屏蔽中断受中断允许标志位 IF 的约束,只有当 IF = 1 时,CPU 才会响应 INTR 请求。如果 IF = 0,即使中断源有中断请求,CPU 也不会响应,这种情况称为中断被屏蔽。在 8086/8088 中断系统中,外部设备的中断请求是通过中断控制器 8259A 来进行统一管理的,由 8259A 决定是否允许一个外设向 CPU 发出中断请求。IBM-PC 机中的可屏蔽中断的中断类型码为 8 ~ 15(08H ~ 0FH),80286 以后的微型计算机还包括 112 ~ 119(70H ~ 77H)。

2. 中断向量表(IVT)

CPU 不论是响应内部中断还是外部中断,都要转向中断服务程序,以便进行相应的中断处理。转向中断服务程序的第一步就是要获得程序的入口地址。在微型计算机系统中,当 CPU 工作在实地址模式下时,服务程序的入口地址存放在一个称为中断向量表(Interrupt

Vector Table, IVT) 的表格中; 在保护模式下则存放在中断描述符表中。8086/8088CPU 只能在实模式下, 其各中断服务程序的入口地址都存放在内存的一个专用区域——中断向量表内。

在 8086/8088 中断系统中, 无论是外部中断还是内部中断, 每个中断源都有一个与之相对应的中断类型码。中断类型码的长度为一个字节, 所以 8086/8088 最多允许处理 256 种类型的中断(中断类型码为 0~255)。CPU 在响应中断时, 首先要得到中断源所对应的中断类型码, 再进一步得到中断服务程序的入口地址。

为了能够根据所得到的中断类型码来找到中断服务程序的入口地址, 8086/8088 系统规定所有中断服务程序的入口地址都存放在中断向量表, 向量表的位置固定在内存的最低 1 KB(即内存中 00000H~003FFH 区域), 共有 256 个表项, 用以存放 256 个中断向量(中断向量就是中断服务程序的入口地址, 亦即服务程序第一条指令的物理地址)。每个中断向量(表项)占 4 B, 其中低位字(2 B)存放中断服务程序入口的偏移地址, 高位字存放中断服务程序入口的段地址。按照中断类型码的大小, 对应的中断向量在中断向量表中有规则地顺序存放, 如图 7.19 所示。

要想获得中断服务程序的入口地址, 只需将中断类型码 n 乘以 4, 所得结果就是中断向量在中断向量表中的存放位置(地址), 即存放中断服务程序入口地址的偏移地址。计算出中断向量地址后, 将 $4n$ 和 $4n+1$ 单元的内容装入 IP, $4n+2$ 和 $4n+3$ 单元的内容装入 CS, 然后就可转入中断服务程序。

例如, 中断类型码为 21H 的中断, 其中断服务程序的入口地址存放在 0000:0084H($4 \times 21\text{H} = 84\text{H}$) 开始的 4 个字节单元中。其中, 0000:0084H 和 0000:0085H 所指单元中的内容为中断子程序入口的偏移地址, 0000:0086H 和 0000:0087H 单元中存放的是中断子程序入口的段地址。

3. 中断处理过程

8086/8088 对各种中断的响应过程是不同的, 主要区别在于如何获得相应的中断类型码。

1) 内部中断响应过程

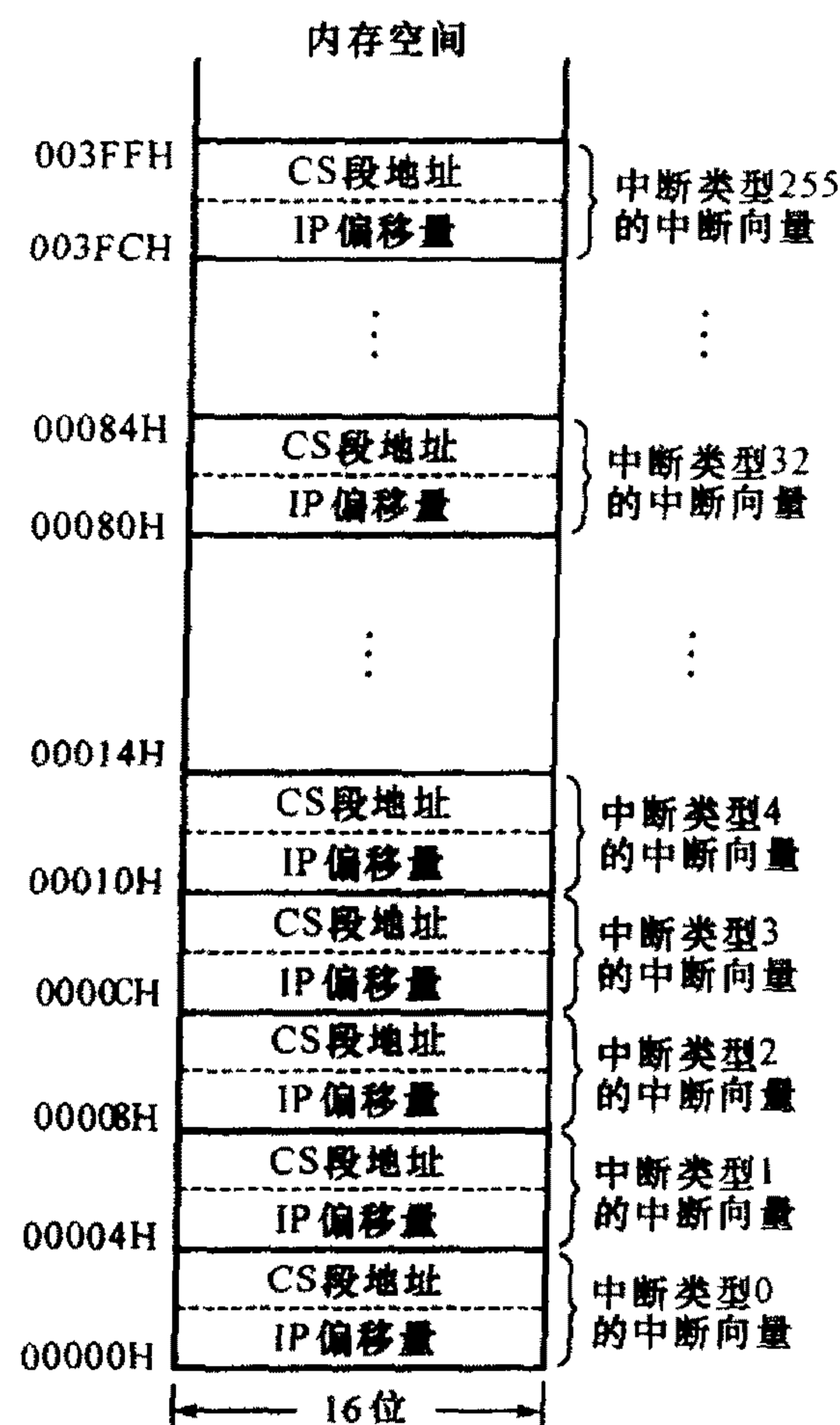


图 7.19 中断向量表结构

CPU 在执行内部中断时,没有中断响应周期。对于除法溢出、单步、断点和溢出中断,中断类型码是自动形成的,而对于 $\text{INT } n$ 指令,其中断类型码由 $\text{INT } n$ 指令中给定的 n 决定。获得中断类型码以后的处理过程顺序为:

- ① 将类型码乘 4,计算出中断向量的地址;
- ② CPU 的标志寄存器入栈,以保护各个标志位,此操作类似于 PUSHF 指令;
- ③ 清除 IF 和 TF 标志,屏蔽新的 INTR 中断和单步中断;
- ④ 保存断点,即把断点处的 IP 和 CS 值压入堆栈,先压入 CS 值,再压入 IP 值;
- ⑤ 根据第一步计算出来的地址从中断向量表中取出中断服务程序的入口地址(段和偏移),分别送至 CS 和 IP 中;
- ⑥ 转入中断服务程序执行。

进入中断服务程序后,首先要保护在中断服务子程序中要使用的寄存器内容,然后进行相应的中断处理,在中断返回前恢复保护的寄存器内容,最后执行中断返回指令 IRET 。 IRET 的执行将使 CPU 按次序恢复断点处的 IP、CS 和标志寄存器,从而使程序返回到断点处继续执行。

内部中断具有如下一些特点:

- ① 中断由 CPU 内部引起,中断类型码的获得与外部无关,CPU 不需要执行中断响应周期去获得中断类型码;
- ② 除单步中断外,内部中断无法用软件禁止,不受中断允许标志 IF 的影响;
- ③ 内部中断何时发生是可以预测的,这有点类似于子程序调用。

2) 外部中断响应过程

(1) 非屏蔽中断响应

NMI 中断不受 IF 标志的影响,也不用外部接口给出中断类型码,CPU 响应 NMI 中断时也没有中断响应周期。CPU 会自动按中断类型码 2 来计算中断服务程序的入口地址,其后的中断处理过程和内部中断一样。

(2) 可屏蔽中断响应

当 INTR 端出现有效的高电平信号时,如果中断允许标志 $\text{IF} = 1$,则 CPU 在当前指令执行完毕后,会产生两个连续的中断响应总线周期。在第一个中断响应总线周期,CPU 将地址/数据总线置高阻,发出第一个中断响应信号 $\overline{\text{INTA}}$ 给中断控制器 8259A,表示 CPU 响应此中断请求,禁止来自其他总线控制器的总线请求。在最大模式时,CPU 还要启动 $\overline{\text{LOCK}}$ 信号,通知总线仲裁器 8289,使系统中其他处理器不能访问总线。在第二个中断响应总线周期,CPU 送出第二个 $\overline{\text{INTA}}$ 信号,该信号通知 8259A 中断控制器将提出请求的中断源的中断类型码放到数据总线上供 CPU 读取。CPU 读取中断类型码 n 后的中断处理过程也和内部中断一样。图 7.20 给出了 8086/8088 对 INTR 的中断响应时序。

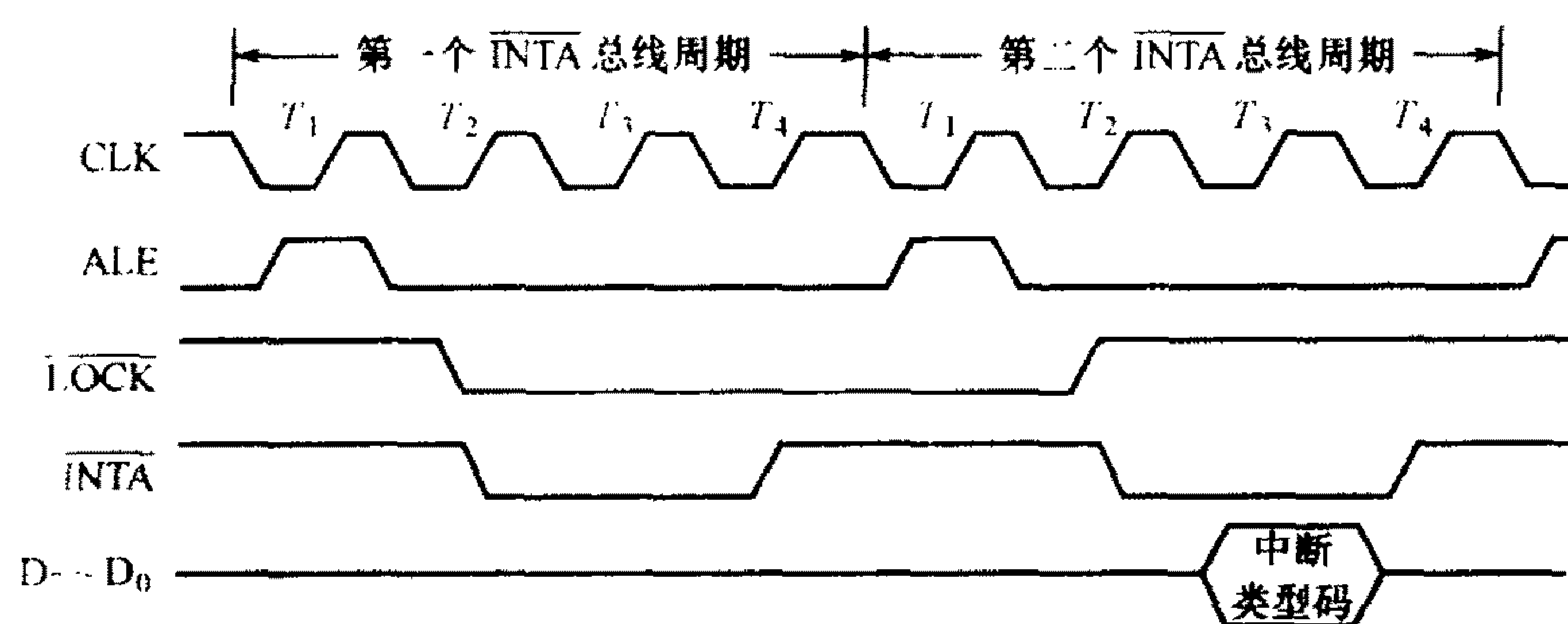


图 7.20 8086/8088 对 INTR 的中断响应时序

在 8086/8088 中断系统中,各类中断的优先级是由 8086/8088 识别中断的前后顺序决定的。在当前指令执行完后,CPU 首先自动查询在指令执行过程中是否有除法出错中断、溢出中断和 $INT\ n$ 中断发生,然后查询 NMI 和 INTR,最后查询单步中断。因此,其中断优先级的顺序依次为软件中断、单步中断、断点中断、非屏蔽中断和可屏蔽中断。8086/8088 中断响应和中断处理流程可以归纳为如图 7.21 所示。

7.3.4 中断程序设计

在微型计算机系统中,响应中断后的断点地址保护、中断控制器的初始化等由硬件系统或操作系统完成,程序员需要做的主要是编写中断服务程序并将中断程序的入口地址放到中断向量表中(也称中断向量设置或初始化中断向量表)。中断服务程序的编写应注意以下几个方面的问题:

- 中断服务程序的格式;
- 保护原中断向量;
- 设置自己的中断向量;
- 恢复原中断向量。

下面简要介绍中断程序设计的一般过程:

(1) 确定要使用的中断类型号

中断类型号不能随便用,有些中断类型号已被系统所占用,若强行使用可能会使系统崩溃。用户程序中应使用那些专门保留给用户的中断类型号。8086/8088 系统中可供用户使用的中断类型号为 60H ~ 66H 和 68H ~ 6FH。

(2) 保存原中断向量

在把自己的中断服务程序的入口地址设置到中断向量表中之前,应先保存该地址中原

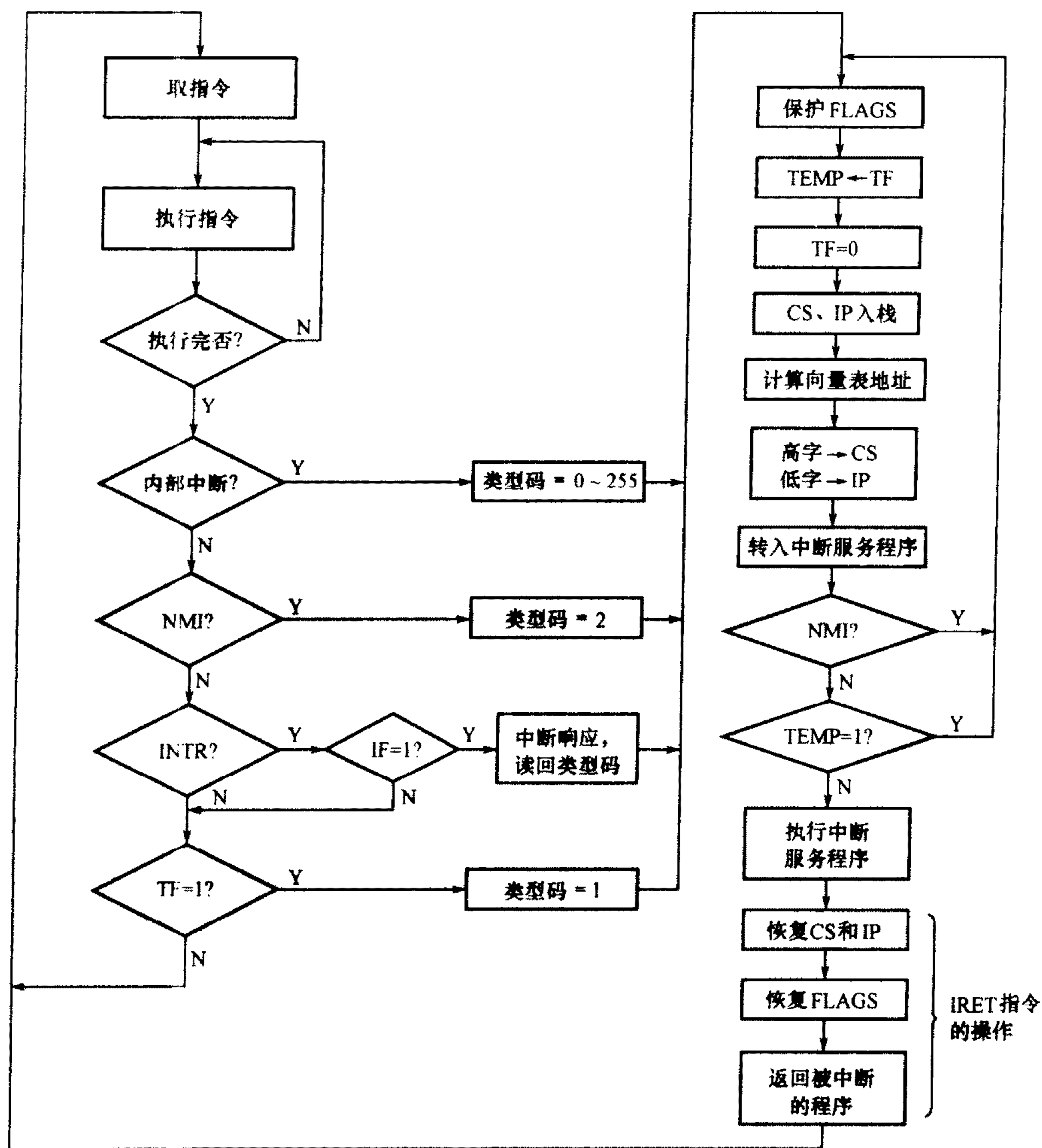


图 7.21 8086/8088 的中断响应和中断处理流程

来的内容。在 DOS 操作系统中有一个包含众多功能模块的功能包,可通过软中断指令 INT 来调用,其中断类型码为 21H,俗称中断 21,或 DOS 功能调用。该功能包中的各功能模块用功能号来区分,其中,功能号为 35H 的功能模块可实现保存原中断向量的工作。35H 号功能可以将任意中断类型码为 n 的中断向量取出,其段地址取到 ES,偏移地址取到 BX。程序段如下:


```
MOV AH,35H
MOV AL,<中断类型码>
INT 21H
```

以上程序执行后,<中断类型码>乘以4指向的中断向量被放在ES:BX中,ES的内容为段地址,BX的内容为偏移地址。取出后的中断向量可进一步保存在用户程序的数据段中,以便退出前恢复。

(3) 设置自己的中断向量

设置自己的中断向量就是把自己编写的中断服务程序的入口地址存入中断向量表的相应表项中。这可以通过直接编程实现,也可利用DOS功能调用的25H号功能模块完成(该方式下的程序格式与上述程序段类似)。以下是直接初始化中断向量表的程序段:

```
MOV AX,0000H
MOV DS,AX                ;中断向量表的段地址送DS
MOV SI,<中断类型码×4>
LEA DX,<中断程序入口>    ;取中断程序入口的偏移地址
MOV [SI],DX              ;中断程序入口的偏移地址送类型码×4所指单元
MOV DX,SEG <中断程序入口> ;取中断程序入口的段地址
MOV [SI+2],DX            ;中断程序入口的段地址送类型码×4+2所指单元
```

(4) 恢复原中断向量

在中断向量设置完成后,如果是硬件中断程序,还应将该硬件中断对应中断控制器的中断屏蔽位开放(详见下节)。然后,打开CPU的中断标志位,以便CPU响应中断。

在中断服务程序退出前一定要恢复原中断向量。这是因为程序一旦退出,该存储区内容将不可预料,若又产生同类型中断,CPU将转移到这个不可预料的内存区去执行,其后果很可能是系统崩溃、死机。

另外,在编写中断服务程序时,要使CPU在中断服务程序中停留的时间越短越好,这就要求中断服务程序要编写得短小精干,能放在主程序中完成的任务,就不要由中断服务程序来完成。

7.3.5 保护模式下的中断响应

8086/8088的中断向量表(IVT)固定设置在00000H~003FFH区域,对于80X86,当其工作在实地址模式下时,中断服务程序的入口地址仍然存放在向量表中,存放入口的偏移地址与8086系统一样,还是用向量码乘以4得到。向量表的位置和大小取决于中断描述符表寄存器IDTR的内容,它在系统初始化时通过LIDT指令加载,使IVT仍然是1KB、起始于内存段地址为0的区域。

80386 以后的微处理器可以工作在保护模式下,此时中断服务程序的入口地址由中断描述符表(IDT)提供。IDT 的位置不再固定,其容量也不再限于 1 KB,每个表项也从 4B 增加到 8B。这里介绍如何利用中断描述符表来确定中断服务程序的入口地址及保护模式下的中断响应过程。

1. 中断描述符表 (Interrupt Descriptor Table)

为了与保护模式下的虚拟存储系统及多任务相适应,IDT 的位置不再固定在内存的最低地址区域,而是被定义成一个特殊的段,其基地址由中断描述符表寄存器 IDTR 决定,可以在内存的任意位置上,中断描述符表由称为门的 8B 描述符组成,这 8B 中包括了 2B 的选择器,4B 的偏移量和 2B 的其他属性。每个门对应一个中断类型,共有 256 个中断类型号,即 IDT 中最多有 256 个描述符。用中断类型码乘以 8 就得到对应的门在 IDT 中的偏移地址。

IDT 的起始地址由中断描述符表寄存器 IDTR(48 位)中的 32 位基地址给出,IDTR 的低 16 位指明 IDT 的界限,即 IDT 的容量(以 B 为单位)。IDTR 的值由 LIDT 指令装入,可见 IDT 的位置和大小都是可变的。保护模式下中断服务程序的入口地址与中断类型码的关系如图 7.22 所示。

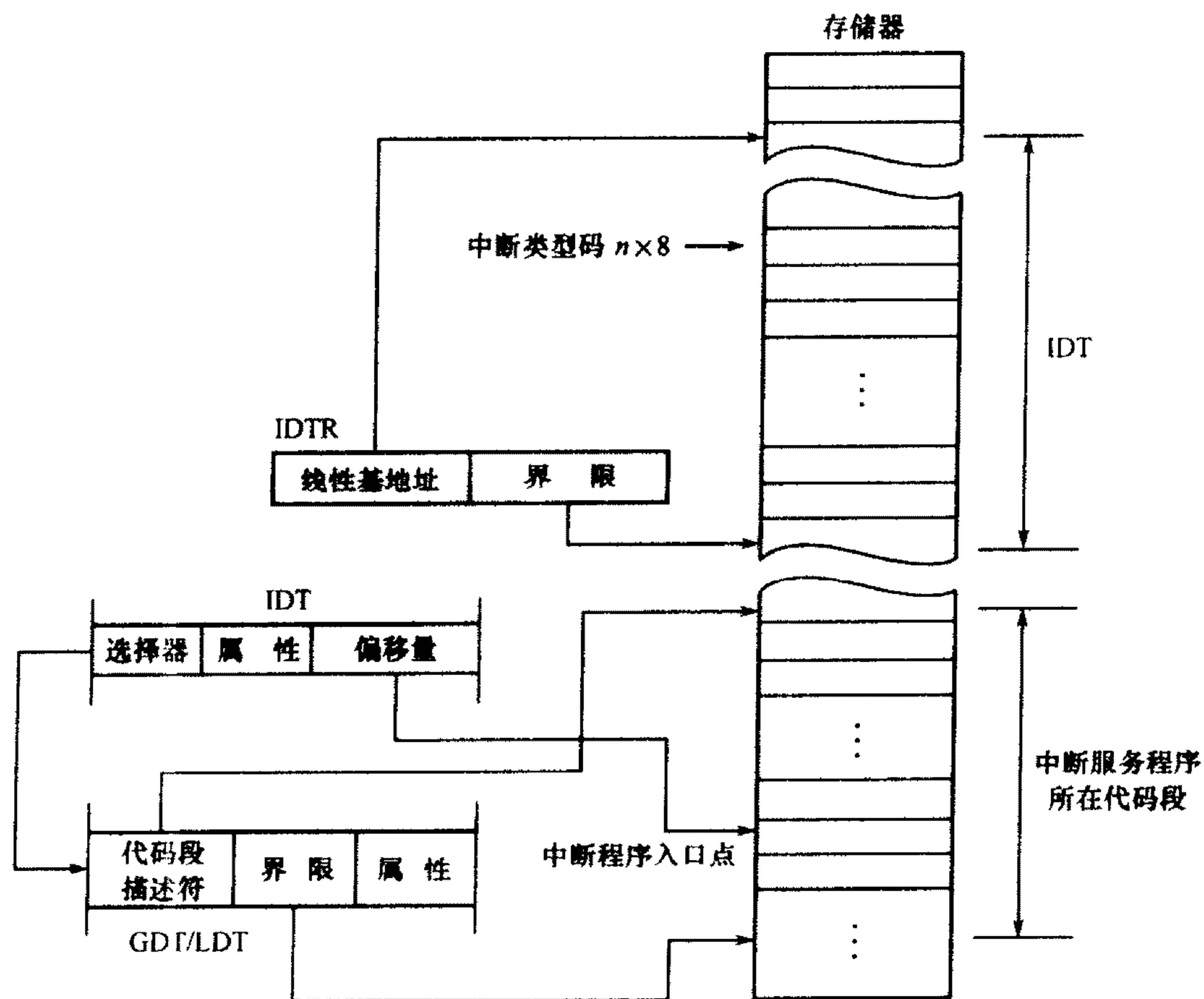
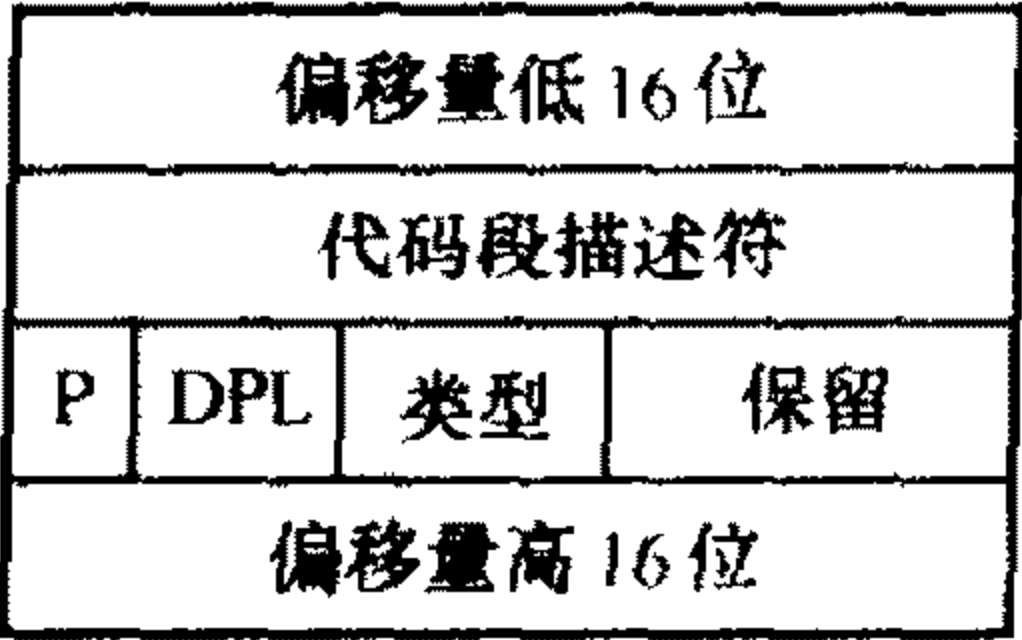


图 7.22 保护模式下中断处理程序入口与中断类型码的关系

IDT 中 8B 的门可分为中断门、陷阱门和任务门。通过中断门和陷阱门处理的中断,其中断服务程序与当前正在执行的程序在同一任务中;而通过任务门处理的中断,其中断服务程序与当前正在执行的程序不在同一任务中。

2. 通过中断门和陷阱门处理的中断

此中断向量码乘以 8 所指向的中断描述符表中的门是中断门或陷阱门,表示要响应的中断服务程序的入口地址由 IDT 的中断门或陷阱门提供。图 7.23 给出了中断门和陷阱门的格式,它包含 2B 的代码段选择符、4B 的段内偏移地址和 2B 其他属性信息。代码段选择符指向全局描述符表 GDT 或局部描述符表 LDT 内的代码段描述符,进而指向一个代码段;4B 的段内偏移地址确定了代码段内的一个具体地址,这就是中断服务程序的入口地址。



类型值:
1110B, 16 位中断门
1111B, 16 位陷阱门

图 7.23 中断门和陷阱门的格式

在找到中断程序的入口之后,下面就可进入中断处理的过程,与实模式类似,也要进行相应的断点保护、执行中断处理程序,在程序执行完后,再执行一条 IRET 指令返回原被中断处。

由中断门或陷阱门处理的中断因与当前任务在同一过程内,其转移的速度比较快,但现场的保护和恢复要由中断服务程序完成。当中断向量码乘以 8 所指向的中断描述符表中的门是任务门时,则表示要执行的中断程序与当前任务不在同一过程,此时因为要进行任务的切换,所需的时间就比较长,但这个切换的过程中包括了现场的保护和恢复。因篇幅所限,有关通过任务门转移的过程不再详述。

7.4 中断控制器 8259A

8259A 是 Intel 公司生产的一种可编程中断控制器 (Programmable Interrupt Controller, PIC),用于对可屏蔽中断请求进行管理,每片可对 8 个中断源实现优先级控制,利用多片 8259A 级联还可扩展至对 64 个中断源实现优先级控制。8259A 可以根据不同的中断源向 CPU 提供不同的中断类型码,还可根据需要对中断源进行中断屏蔽。另外,通过编程选择,还可使其工作在多种方式下,以适应不同应用场合的需要。

7.4.1 8259A 的引脚及内部结构

1. 外部引脚

8259A 共有 28 根引出线,其外部引出线定义如图 7.24 所示。

D₀ ~ D₇ 双向数据线,与系统的数据总线相连。在编写程序时将控制字、命令字由此写入;中断响应时,中断向量码由此送给 CPU。

$\overline{\text{WR}}$ 、 $\overline{\text{RD}}$ 写和读控制信号,与系统总线的 $\overline{\text{IOW}}$ 、 $\overline{\text{IOR}}$ 相连接。

$\overline{\text{CS}}$ 片选信号,当 $\overline{\text{CS}}$ 为低电平时,8259A 被选中,CPU 才能对它进行读/写操作。此引出线连到系统的 I/O 译码器输出,由此确定 8259A 在系统 I/O 地址空间的基地址。

A₀ 8259A 内部寄存器的选择信号。它与 $\overline{\text{CS}}$ 、 $\overline{\text{WR}}$ 、 $\overline{\text{RD}}$ 信号相配合,对不同的内部寄存器进行读/写。当 8259A 与 8086 和 80286 相连时,A₀ 与地址总线的 A₁ 端连接。若 8259A 的 A₀ = 0,表示 CPU 对其操作针对的是偶数地址;A₀ = 1,则表示针对的是奇数地址。

INT 8259A 的中断请求输出信号,可直接接到 CPU 的 INTR 输入端。

$\overline{\text{INTA}}$ 中断响应输入信号。在中断响应过程中 CPU 的中断响应信号由此端进入 8259A。

CAS₀ ~ CAS₂ 级联控制线。当多片 8259A 级联工作时,利用一片作为主控芯片,其他作为从属芯片。对主控 8259A,其 CAS₀ ~ CAS₂ 为输出;各从片的 CAS₀ ~ CAS₂ 为输入。主片的 CAS₀ ~ CAS₂ 与从片的 CAS₀ ~ CAS₂ 对应相连。当某从片 8259A 提出中断请求时,主片 8259A 通过 CAS₀ ~ CAS₂ 送出相应的编码给从片,使从片的中断被允许。

$\overline{\text{SP/EN}}$ 双功能引出线。当 8259A 工作在缓冲模式时,它为输出,用以控制缓冲器的传送方向。当数据从 CPU 送往 8259A 时, $\overline{\text{SP/EN}}$ 输出为高电平;当数据从 8259A 送往 CPU 时, $\overline{\text{SP/EN}}$ 输出为低电平。在 8259A 工作在非缓冲模式时,它为输入,用于指定 8259A 是主片还是从片, $\overline{\text{SP}} = 1$ 的 8259A 为主片, $\overline{\text{SP}} = 0$ 的 8259A 为从片。在只有一片 8259A 时,它接高电平。

IR₀ ~ IR₇ 中断请求输入信号,与外设的中断请求线相连。上升沿或高电平(可通过编程设定)时表示有中断请求到达。

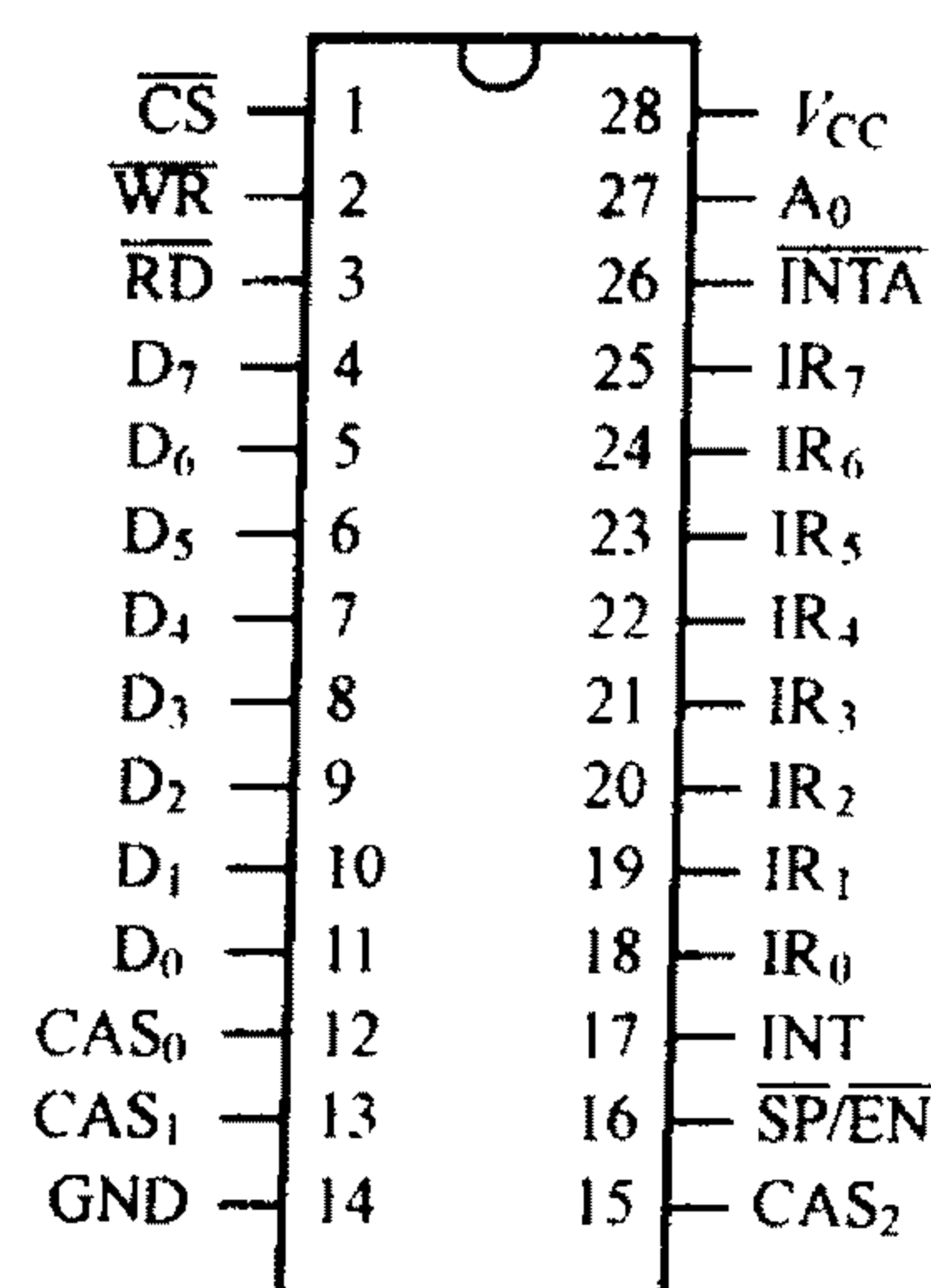


图 7.24 8259A 引脚排列图

2. 8259A 的内部结构

8259A 的内部结构如图 7.25 所示。它主要由以下几个部分组成:

- ① 中断请求寄存器 IRR(Interrupt Request Register);
- ② 中断服务寄存器 ISR(In - Service Register);
- ③ 中断屏蔽寄存器 IMR(Interrupt Mask Register);
- ④ 中断优先级判别电路;

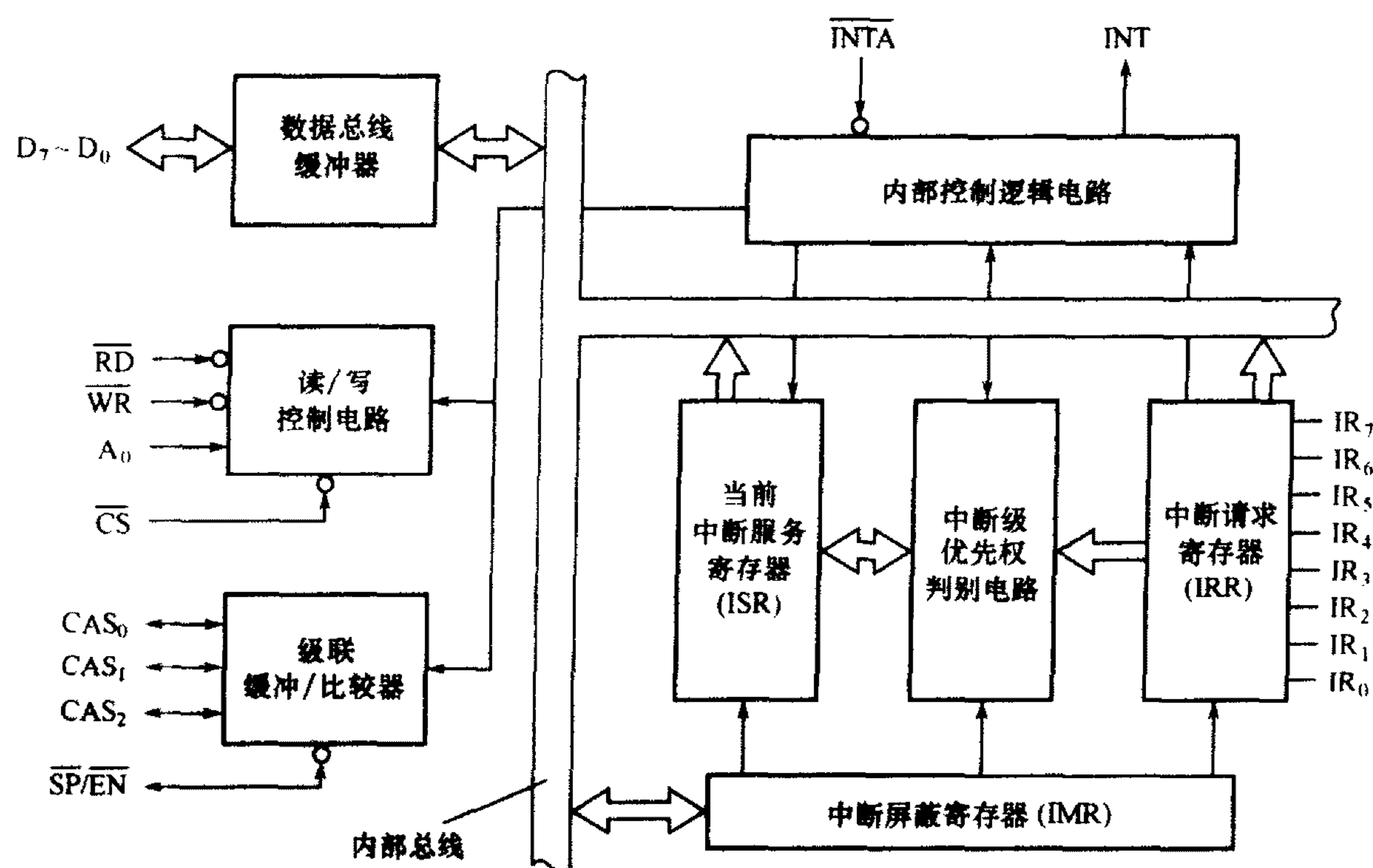


图 7.25 8259A 内部结构框图

- ⑤ 数据总线缓冲器；
- ⑥ 读/写控制电路；
- ⑦ 控制逻辑和级联缓冲/比较器。

1) 中断请求寄存器 IRR

IRR 负责保存从 $IR_0 \sim IR_7$ 来的中断请求信号。某 1 位为 1, 则表示相应引出线上有中断请求产生。请求信号至少应保持到该请求被响应为止。中断响应后, 该 IR 输入线上的请求信号应撤销, 否则, 在中断处理完结后, 该 IR 线上的高电平可能会引起又一次中断服务。

2) 中断服务寄存器 ISR

ISR 为 8 位寄存器 ($IS_0 \sim IS_7$ 分别对应 $IR_0 \sim IR_7$), 用于保存所有正在服务的中断源。在中断响应时, 判优电路把发出中断请求的中断源中优先级最高的中断源所对应的位设置为 1, 以表示该中断请求正在处理中。ISR 的某 1 位 IS_i 置 1 可阻止与它同级及更低优先级的请求被响应, 但不阻止比它优先级高的中断请求被响应, 即允许中断嵌套。所以, ISR 中可能有不止 1 位被置 1。当 8259A 收到中断结束 (End Of Interrupt, EOI) 命令时, ISR 相应位会被清除。对自动 EOI 操作 (Automatic EOI, AEOI), ISR 寄存器中刚被置 1 的位在中断响应结束时自动复位。

3) 中断屏蔽寄存器 IMR

IMR 用于存放中断屏蔽字,它的每一位分别与 $IR_7 \sim IR_0$ 相对应。其中,为 1 的位所对应的中断请求将被屏蔽;为 0 的位所对应的中断请求输入不受影响。

4) 中断判优电路

中断判优电路监测从 IRR、ISR 和 IMR 来的输入,并确定是否应向 CPU 发出中断请求。在中断响应时,它要确定 ISR 寄存器哪 1 位应置 1,并将相应的中断类型码送给 CPU。在 EOI 命令时,它要决定 ISR 寄存器哪 1 位应复位。

7.4.2 8259A 的工作原理

1. 8259A 的工作过程

由于 8259A 是可编程的中断控制芯片,故在进入正常工作前首先应对其进行初始化,也就是由 CPU 执行一段程序,向 8259A 写入若干控制字,使其处于指定的工作方式。当初始化完成后,8259A 就处于就绪状态,随时可接受外设送来的中断请求信号。当外设发出中断请求后,8259A 对外部中断请求的处理过程如下:

① 当有一条或若干条中断请求输入线($IR_0 \sim IR_7$)上的中断请求信号有效时,则 IRR 的相应位置 1。

② 若中断请求线中至少有一条是中断未被屏蔽的,则 8259A 由 INT 引出线向 CPU 发出中断请求信号 INTR。

③ 若 CPU 是处于开中断状态,则在当前指令执行完以后,CPU 用 \overline{INTA} 信号作为对 INTR 的响应。

④ 8259A 在接收到 CPU 发出的第一个 \overline{INTA} 脉冲后,使最高优先级的 ISR 位置 1,并使相应的 IRR 位复位。

⑤ 在第二个中断响应总线周期中,CPU 再输出一个 \overline{INTA} 脉冲,这时 8259A 就把刚才选定的中断源所对应的 8 位中断类型码放到数据总线上。CPU 读取该中断类型码并乘以 4,就可以从中断向量表中取出中断服务程序的入口地址并转去执行。

⑥ 若 8259A 工作在自动中断结束 AEIOI 方式,在第二个 \overline{INTA} 脉冲结束时,就会使中断源所对应的 ISR 中的相应位复位。对于非自动中断结束方式,则由 CPU 在中断服务程序结束时向 8259A 写入 EOI 命令,才能使 ISR 中的相应位复位。

2. 8259A 的工作方式

8259A 具有非常灵活的中断管理方式,可以通过编程来设置(编程方法将在其后逐步介绍)其工作在哪一种方式下。

1) 中断优先方式

8259A 具有两种优先级控制方式,即固定优先级和循环优先级。

(1) 固定优先级方式

在此方式下,只要不重新设置优先级别,所有中断请求的中断优先级就是固定不变的。8259A 加电后就处于这种方式,刚加电时,默认 IR_0 优先级最高(0 级为最高级), IR_7 优先级最低(7 级为最低级),这种优先顺序也可通过程序予以改变,使它按另外一种顺序排列。图 7.26 给出了两种固定优先级的顺序。

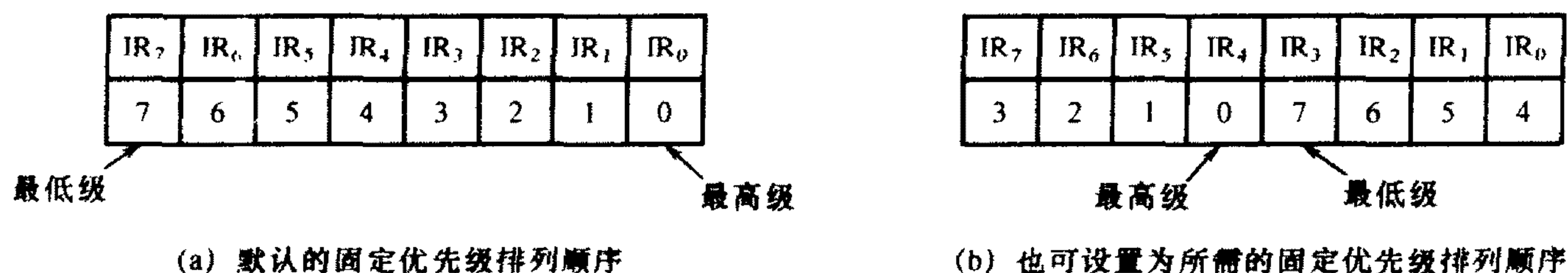


图 7.26 固定优先级方式

(2) 循环优先级方式

在实际应用中,许多中断源的优先级别是一样的,若固定了优先级,则低级别的中断源的中断请求有可能总是得不到服务。解决的方法是使这些中断源轮流处于最高优先级。这就是自动循环中断优先级方式。

在自动循环优先级方式中,优先级顺序是变化的。一个中断源得到中断服务以后,它的优先级自动降为最低,原来比它低一级的中断则为最高级,依次排列。例如,若初始优先级从高到低依次为 $IR_0, IR_1, IR_2, \dots, IR_7$ 。如果此时 IR_4 和 IR_6 有中断请求,则先处理 IR_4 。在 IR_4 被服务以后, IR_4 自动降为最低优先级, IR_5 成为最高优先级,这时中断源的优先级顺序变为 $IR_5, IR_6, IR_7, IR_0, IR_1, IR_2, IR_3, IR_4$ 。

2) 中断嵌套方式

无论是固定优先级方式还是自动循环优先级方式,它们都允许中断嵌套,即允许更高优先级的中断打断当前的中断处理过程。8259A 允许两种中断嵌套方式。

(1) 一般全嵌套方式

一般全嵌套方式是 8259A 最常用的工作方式,简称为全嵌套方式。它的核心是在执行中断服务程序的过程中,只允许响应比该中断源优先级高的中断请求,而与它同级或优先级更低的中断源的申请将被屏蔽。

(2) 特殊全嵌套方式

特殊全嵌套方式和一般全嵌套方式的差别在于:在特殊全嵌套方式下,当处理某一级中断时,如果有同级的中断请求,8259A 也会给予响应,从而实现一个中断处理过程能被另一个具有同等级别的中断请求所打断。

特殊全嵌套方式一般用在 8259A 级联的系统中。在这种情况下,只有主片 8259A 允许编程为特殊全嵌套方式。这样,当来自某一从片的中断请求正在处理时,主片除对来自优先级较高的本片上其他 IR 引出线上的中断请求进行开放外,同时对来自同一从片的较高优先级请求也会开放。使主片不封锁得到响应的从片的 INT 输入,以便让从片上优先级别更高的中断能够得到响应。比如在图 7.27 中,如果当前正在处理的中断请求是由从片的 IR_3 端接入,若主片设置为一般嵌套方式,则从片的 $IR_0 \sim IR_2$ 端连接的中断请求都会被主片视为与当前正在处理的中断源优先级相同(因为该从片的所有中断请求都是通过主片的 IR_4 端输入 CPU 的),从而不会被响应;但如果主片设置为特殊嵌套方式,从片 $IR_0 \sim IR_2$ 的中断请求就不会被封锁,而只封锁从片 $IR_4 \sim IR_7$ 及主片 $IR_5 \sim IR_7$ 的中断请求。

另外,在特殊全嵌套方式中,中断结束的操作必须用软件检查刚结束的中断是否为从片的惟一中断。其方法是:先向从片发一正常结束中断命令 EOI,然后读 ISR 内容。若为 0 表示只有一个中断服务,这时再向主片发一个 EOI 命令;否则,说明该从片有两个以上中断,则不应向主片发 EOI 命令,待该从片中断服务全部结束后,再发送 EOI 命令给主片。

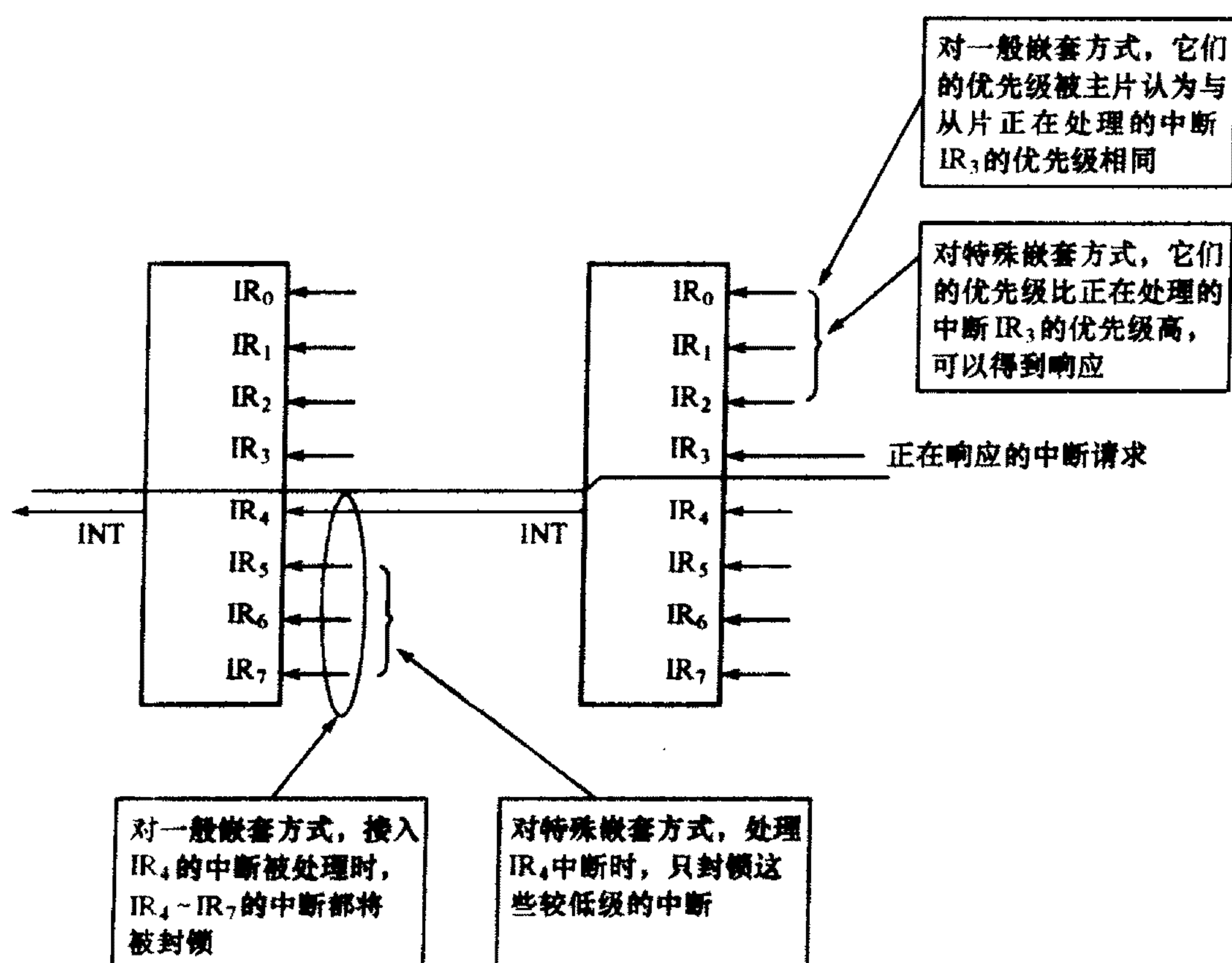


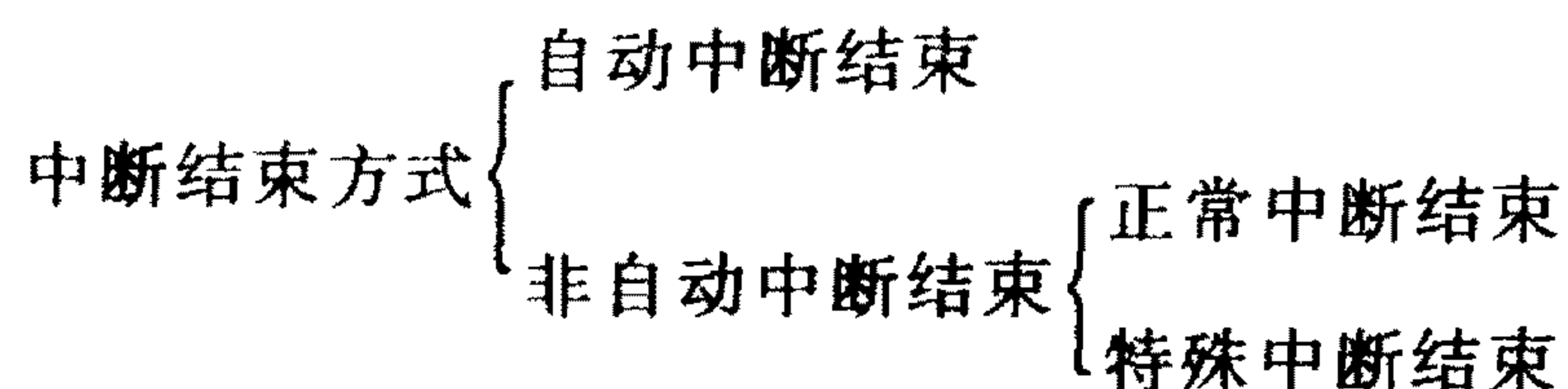
图 7.27 两种中断嵌套方式下的响应过程

3) 中断结束处理方式

所谓的中断结束是指 8259A 结束中断的处理,而不是 CPU 结束执行中断服务程序。不

论采用哪一种优先级方式工作,当一个中断请求得到响应时,8259A 会对中断服务寄存器 ISR 中相应位 IS 置 1。而当中断服务程序结束时,则必须将该 IS 位清零,否则,8259A 的中断控制功能就会不正常。这个使 IS 位复位的动作就是中断结束处理。

8259A 的中断结束方式可分为以下两种:



(1) 自动中断结束方式(AEOI)

若采用 AEOI 方式,则在第二个中断响应周期的 \overline{INTA} 信号的后沿,8259A 会自动把中断服务寄存器 ISR 中的对应位清零。这样,尽管系统正在为某个设备进行中断服务,但对 8259A 来说,中断服务寄存器中却没有保留正在服务的中断状态。所以,对 8259A 来说,好像中断服务已经结束了一样。这种最简单的中断结束方式,只能用于没有中断嵌套的情况。

(2) 正常中断结束方式

正常中断结束方式又称一般中断结束方式,它可以配合全嵌套优先级工作方式使用。当 CPU 用输出指令往 8259A 发出正常中断结束 EOI 命令时,8259A 就会把 ISR 中已置 1 的位中的最高位复位。因为在全嵌套方式中,置 1 的最高 ISR 位对应了最后一次被响应的和被处理的中断,也就是当前正在处理的中断,所以,把已置 1 的位中最高的 ISR 位复位相当于结束了当前正在处理的中断。

(3) 特殊中断结束方式(SEOI)

在非全嵌套方式下,由于中断优先级不断改变,无法确定当前正在处理的是哪一级中断,这时就要采用特殊中断结束方式。这种方式反映在程序中就是要发一条特殊中断结束命令,这个命令中指出了要清除 ISR 中的哪一位。

要注意的是,不管是正常中断结束方式,还是特殊中断结束方式,在一个中断服务程序结束时,对于级联使用的 8259A 都必须发两次中断结束命令,一次发给主片,另一次发给从片。

4) 中断源屏蔽方式

屏蔽中断源就是禁止某个中断源的中断请求。8259A 的 8 个中断请求都可根据需要单独屏蔽,屏蔽是通过编程使得中断屏蔽寄存器 IMR 相应位置 0 或 1,从而允许或禁止该位所对应的中断。8259A 有两种屏蔽方式。

(1) 普通屏蔽方式

在普通屏蔽方式中,将 IMR 某位置 1,则它对应的中断请求就被屏蔽,从而使这个中断请求不能从 8259A 送到 CPU。如果该位置 0,则表示允许该中断请求传送给 CPU。

(2) 特殊屏蔽方式

在一般情况下,不允许低优先级中断源打断正在执行的较高优先级的中断服务程序。但在特殊屏蔽方式下,允许除当前正在处理的中断以外的所有中断请求被响应。其工作过程为:在芯片初始化时首先将屏蔽方式设置为特殊屏蔽方式,再编程将正在处理的中断源屏蔽掉(将IMR中对应的位置1)。这样,除了正在服务的这级中断被屏蔽(不允许产生进一步中断)外,其他各级中断全部被开放。

由于特殊中断屏蔽方式打乱了正常的全嵌套结构,被处理的程序不见得是当前优先级最高的事件,所以不能用正常EOI命令来使其ISR位复位。但在退出SMM方式之后,仍可用正常EOI命令来结束中断服务。

5) 中断触发方式

外设的中断请求信号从8259A的引出线IR端引入,根据实际需要,IR端的中断触发方式有两种。

(1) 边沿触发方式

当某个IR引出线上出现上升沿信号时表示有中断请求,高电平并不表示有中断请求。

(2) 电平触发方式

某个IR引出线上出现高电平表示有中断请求。这种方式下,应注意及时撤除高电平,否则可能引起不应该有的第二次中断。

无论是边沿触发还是电平触发,中断请求信号IR都应维持足够的宽度。即在第一个中断响应信号 \overline{INTA} 结束之前,IR都必须保持高电平。如果IR信号提前变为低电平,8259A就会自动假设这个中断请求来自引出线 IR_7 。这种办法能够有效地防止由IR输入端上严重的噪声尖峰而产生的中断。为实现这一点,对应 IR_7 的中断服务程序可只执行一条返回指令,从而滤除这种中断。但如果 IR_7 另有它用,仍可通过读ISR状态来识别非正常的 IR_7 中断。因为正常的 IR_7 中断会使ISR的 IS_7 位置位,而非正常的 IR_7 中断不会使ISR的 IS_7 位置位。

6) 级联工作方式

如果系统中的中断源超过8个,就无法用一片8259A来进行管理,这时可采用8259A的级联工作方式。指定一片8259A为主控芯片(主片),它的INT接到CPU上。而其余的8259A芯片均作为从属芯片(从片),其INT输出分别接到主控芯片的IR输入端。由于8259A有8个IR输入端,故一个主控8259A可以连接8片从属8259A,最多允许有64个IR中断请求输入。

由一片主控8259A和两片从属8259A构成的级联中断系统如图7.28所示。图中3个8259A均有各自的地址,由 \overline{CS} 和A0来决定。主片8259A的 $CAS_0 \sim CAS_2$ 作为输出连接到从片的 $CAS_0 \sim CAS_2$ 上,而两个从片的INT分别接主控芯片的 IR_3 和 IR_6 。图中省略了 \overline{CS} 译码器。

在级联系统中,每一片8259A,不管是主片还是从片,都有各自独立的初始化程序,以便

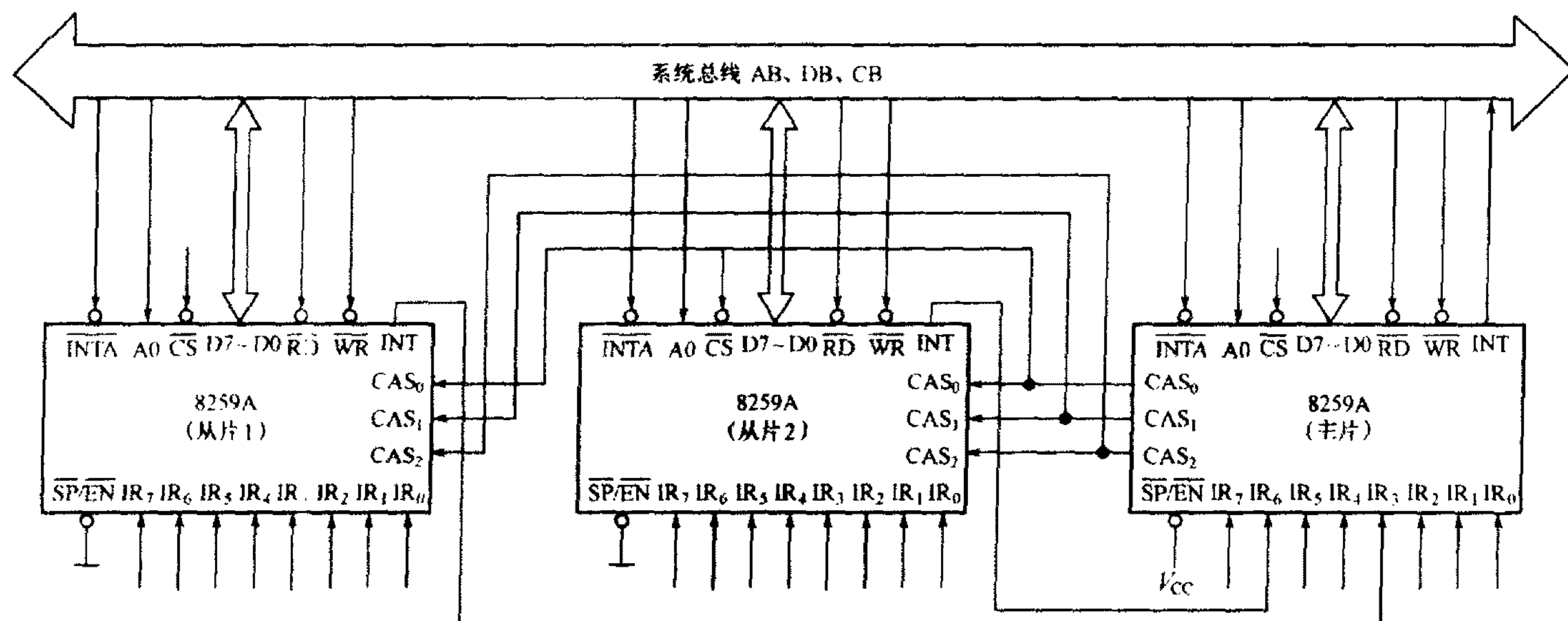


图 7.28 8259A 级联工作方式示意图

设置各自的工作状态。在中断结束时要连发两次 EOI 命令,分别使主片和相应的从片完成中断结束操作。中断响应时,若中断请求是来自于从片的 IR,则中断响应时主片 8259A 会通过 $CAS_0 \sim CAS_2$ 来通知相应的从片 8259A,而从片 8259A 即可把 IR 对应的中断向量码放到数据总线上。

在级联方式下,可采用前面提到的特殊全嵌套方式,以允许从片上优先级更高的 IR 产生中断。在将主控片初始化为特殊全嵌套方式后,从片的中断响应结束时,要用软件来检查中断状态寄存器 ISR 的内容,看看本从片上还有无其他中断请求未被处理。如果没有,则连发两个 EOI,使从片及主片结束中断。若还有其他未被处理的中断,则应只向从片发一个 EOI 命令,而不向主片发 EOI 命令。

7.4.3 8259A 的命令字

8259A 的命令字分为两类:

① 初始化命令字 用于设置 8259A 的初始状态。在系统加电之后,通过写入初始化命令字 ICW(Initialization Command Word),使其处于准备就绪状态;

② 操作方式命令字 由 CPU 向 8259A 送 3 个字节的操作命令字 OCW(Operation Command Word),可实现对工作状态、中断方式及中断响应次序等的管理。OCW 可在 8259A 初始化以后的任何时刻写入。

初始化命令字 ICW 和操作命令字 OCW 在写入 8259A 后,将被保存在内部的 ICW 和 OCW 寄存器组中。

1. 初始化命令字

8259A 共有 4 个初始化命令字: ICW_1 、 ICW_2 、 ICW_3 和 ICW_4 。各命令字的格式分别如下:

1) ICW_1

ICW_1 命令字的格式及各位的含义如图 7.29 所示,其中,X 位表示无关位,可置为 0。

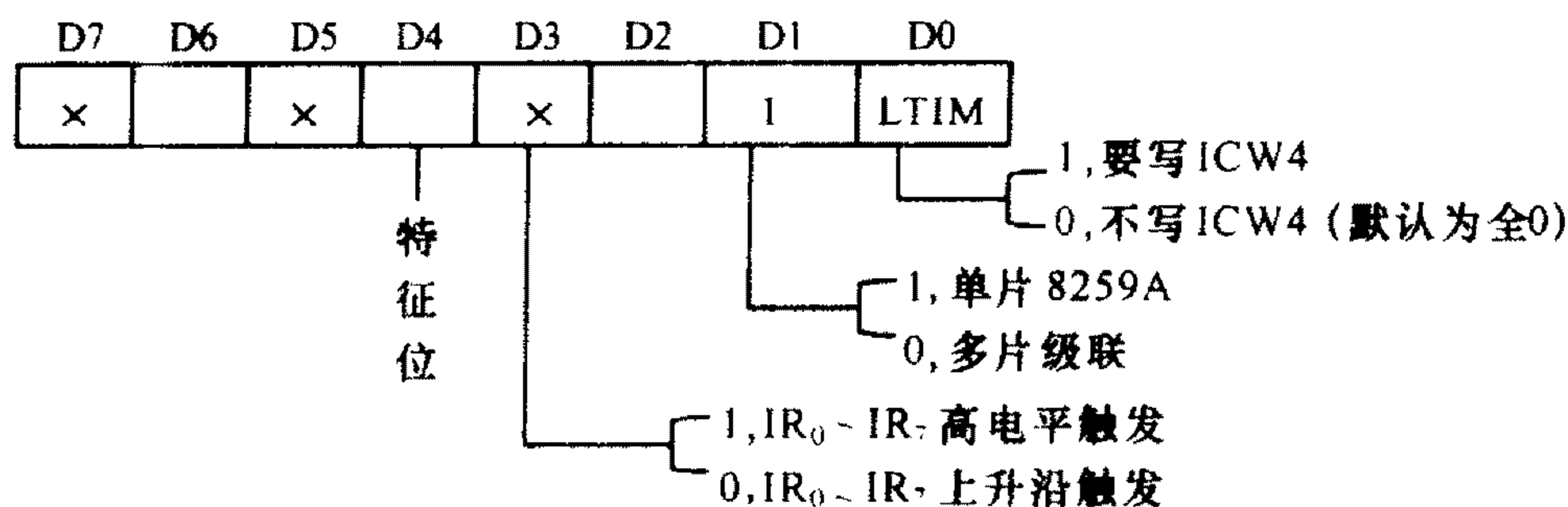


图 7.29 ICW_1 命令字格式

写 ICW_1 的条件: $A_0 = 0, D_4 = 1$ 。这时写入的数据就被当成 ICW_1 。写 ICW_1 意味着重新初始化 8259A。在写 ICW_1 的同时,8259A 还做以下几项工作:

- ① 清除 ISR 和 IMR;
- ② 将中断优先级设成初始状态: IR_0 (最高)— IR_7 (最低);
- ③ 设定为普通屏蔽方式;
- ④ 采用非自动 EOI 中断结束方式;
- ⑤ 状态读出电路预置为读 IRR。

2) ICW_2

ICW_2 用于设置中断向量码,其格式如图 7.30 所示。它的最低 3 位对应 8259A 的 8 个中断请求输入端的序号,高 5 位在初始化编程时由用户确定。当 CPU 响应中断时,在第二个 INTA 负脉冲期间,8259A 将设定的中断类型码送上数据总线。

例如,IBM PC 机中的 ICW_2 被初始化为 08H,即 IR_0 的中断向量码为 08H, IR_7 的中断向量码为 0FH 等。

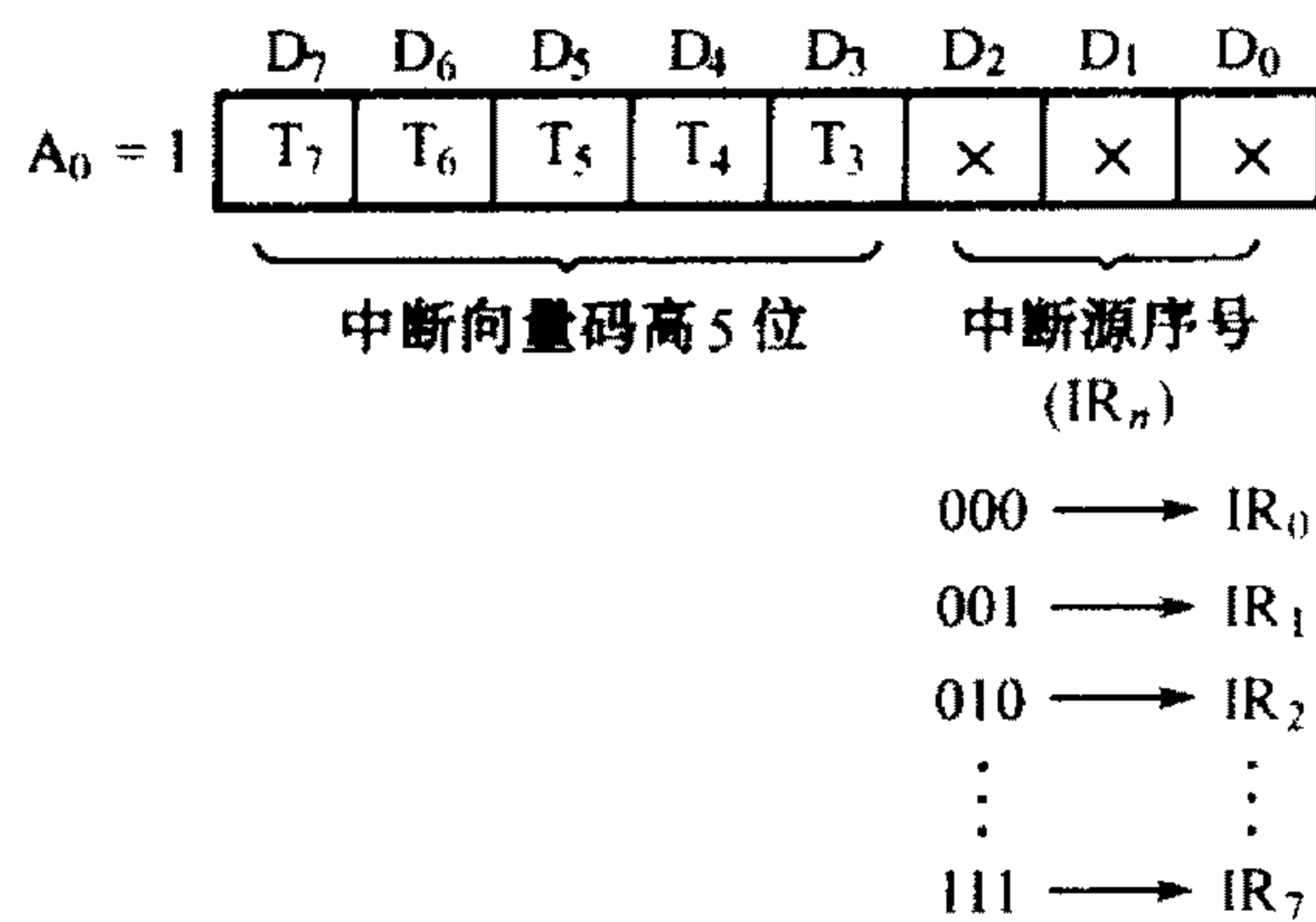


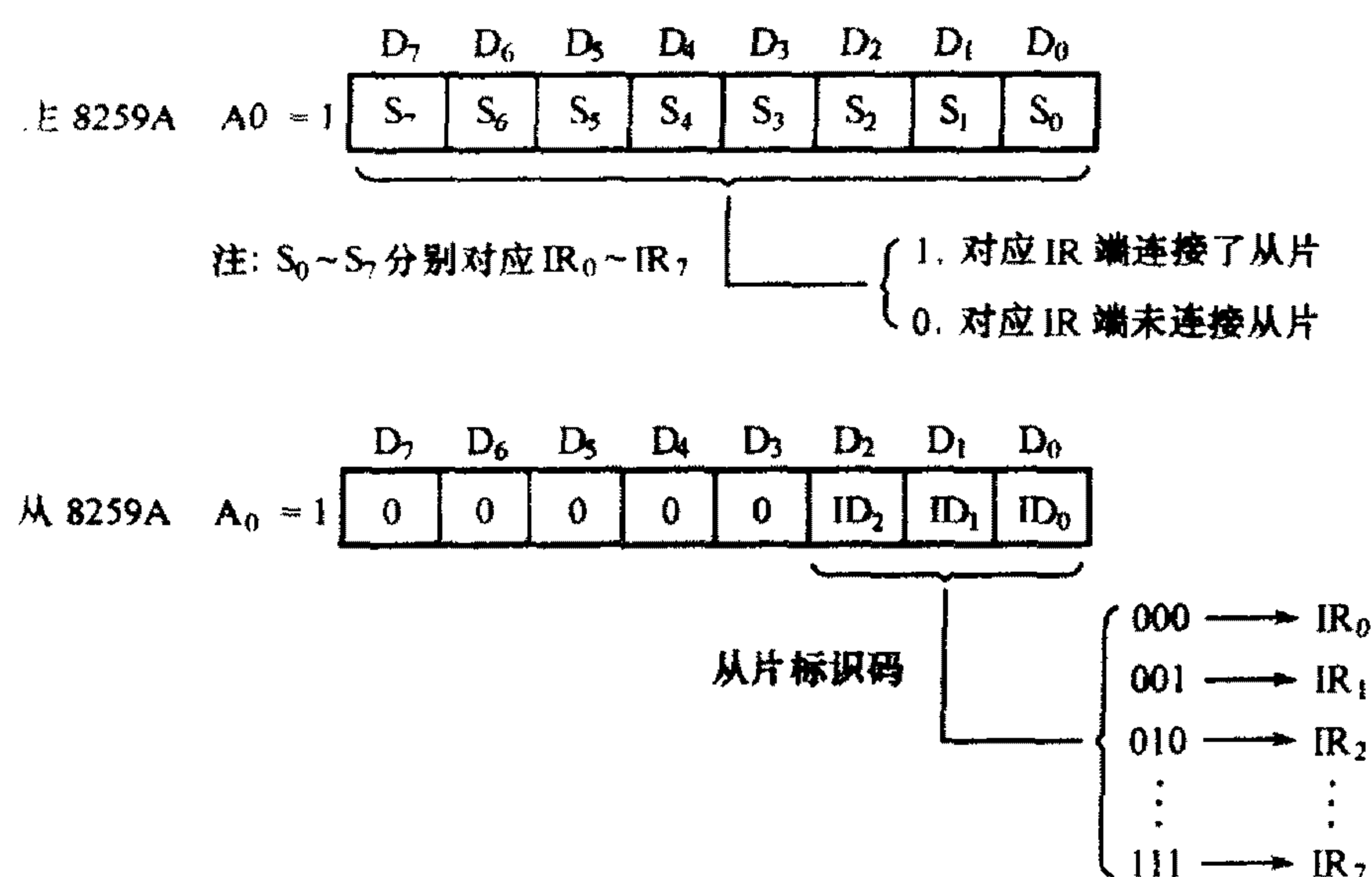
图 7.30 ICW_2 命令字格式

在写 ICW_2 时,地址信号 $A_0 = 1$ 。

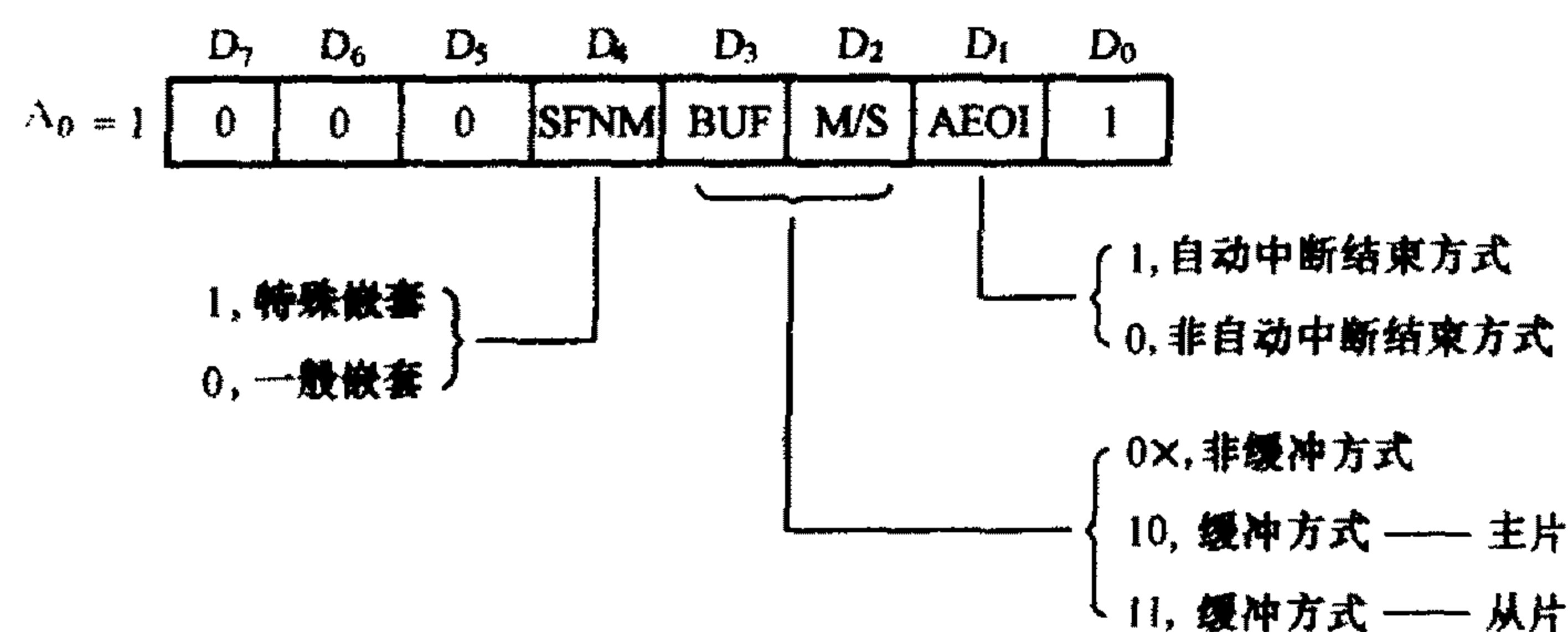
3) ICW_3

ICW_3 命令字仅在多片 8259 级联时需要写入。主片 8259A 的 ICW_3 与从片 ICW_3 的格式

不同。主控 8259A 的 ICW₃ 命令字用于说明其上的哪一个中断请求端 IR 与从片的 INT 端相连,相应地,从属 8259A 的 ICW₃ 命令字中的低 3 位代码说明了本片的 INT 端与主片的哪个 IR 端连接。两种 ICW₃ 的格式分别如图 7.31 所示。

图 7.31 ICW₃ 命令字4) ICW₄

ICW₄ 用于设置中断结束方式,其格式如图 7.32 所示。

图 7.32 ICW₄ 命令字格式

图中的缓冲方式是指 8259A 工作于级联方式时,为增大驱动能力,在其数据线与系统总线之间增加一级缓冲器。这时,8259A 将 $\overline{SP}/\overline{EN}$ 作为输出端,输出一个允许信号,用来控制缓冲器的打开与关闭。而主片与从片只能用 D₂ (M/S 位) 来区分 (主片 = 1, 从片 = 0)。在非缓冲方式时,若 8259A 工作在级联方式, $\overline{SP}/\overline{EN}$ 引脚为输入端,用来区分主片 (高电平) 和从片 (低电平)。

从以上对 8259A 的 4 个初始化命令字格式的说明中可以发现,在写 ICW₂、ICW₃ 和 ICW₄ 时,I/O 地址都为奇数($A_0 = 1$),即它们的写入地址相同。那么如何区分当前写入的到底是哪个命令字呢?对这一点,8259A 是靠写入顺序来区分的,即芯片在初始化时必须严格按照图 7.33 所示的 ICW₁—ICW₂—ICW₃—ICW₄ 这样的写入顺序进行。

2. 操作命令字

操作命令字用于改变 8259A 的中断控制方式、在需要时屏蔽某些中断源以及读出 8259A 的工作状态信息(IRR、ISR、IMR)。操作命令字在初始化完成后任意时刻均可写入,写入顺序没有严格要求。但端口地址有严格规定,OCW₁ 必须写入奇地址端口($A_0 = 1$),OCW₂ 和 OCW₃ 必须写入偶地址端口($A_0 = 0$)。

1) OCW₁

OCW₁ 是中断屏蔽字,用于决定哪些中断请求线被屏蔽。初始状态各位全为 0(全部允许中断)。OCW₁ 的格式如图 7.34 所示。

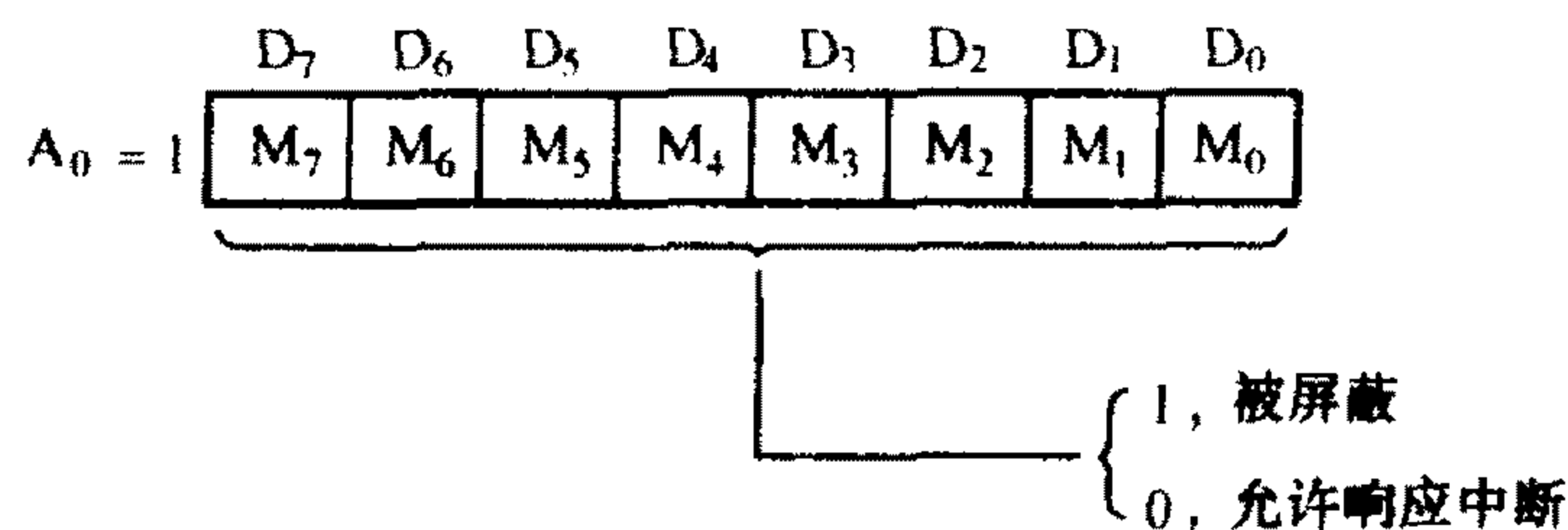


图 7.34 OCW₁ 操作命令字格式

2) OCW₂

OCW₂ 为中断结束和优先级循环命令字,其作用是对 8259A 发出中断结束命令 EOI,并且可以控制中断优先级的循环。OCW₂ 的格式如图 7.35 所示。它与 OCW₃ 共用一个端口地址,但其特征位 $D_4D_3 = 00$,因此不会发生混淆。OCW₂ 各位的含义如下:

R 优先级循环控制位。当 $R = 0$ 时,表示使用固定优先级,IR₇ 最低,IR₀ 最高;当 $R = 1$ 时,表示使用循环优先级。一个优先级级别的中断服务结束后,它的优先级就变为最低级,而下一个优先级变为最高级。

SL 特殊循环控制。当 $SL = 1$ 时,使 $L_2 \sim L_0$ 对应的 IR_i 为最低优先级。SL = 0 时, $L_2 \sim L_0$

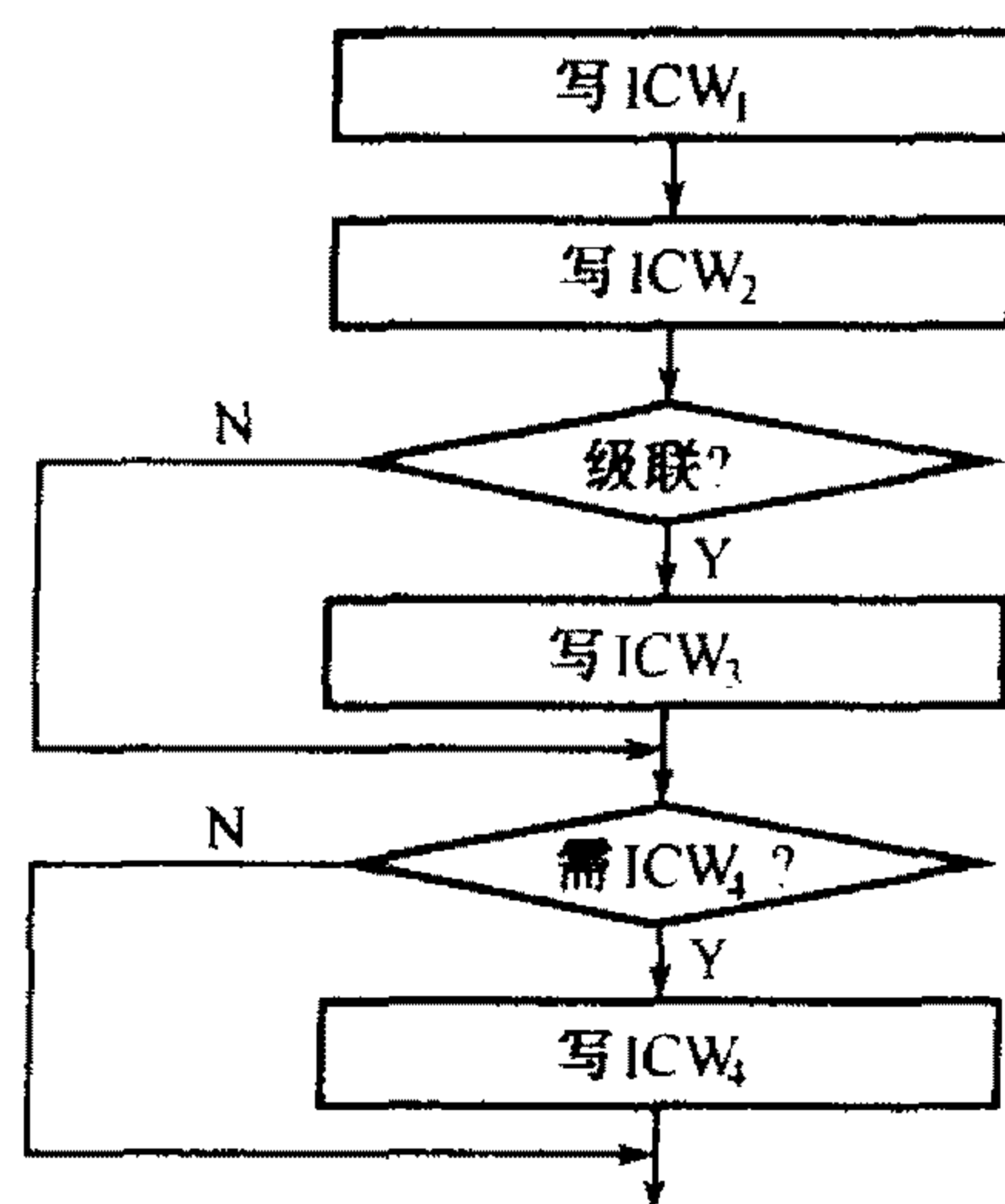


图 7.33 8259A 的初始化顺序

的编码无效。

EOI 是中断结束命令。该位为 1 时,则复位现行中断的 ISR 中的相应位,以便允许 8259A 再为其他中断源服务。在 ICW₄ 的 AEOI = 0(非自动 EOI)的情况下,需要用 OCW₂ 来复位现行中断的 ISR 中的相应位。

L₂ ~ L₀ 第一个作用是设定哪个中断优先级最低,用来改变 8259A 复位后所设置的默认优先级别。第二个作用是在特殊中断结束命令中指明 ISR 的哪一位要被复位。

R、SL、EOI 3 位组合所代表的意义见图 7.35 中的说明。

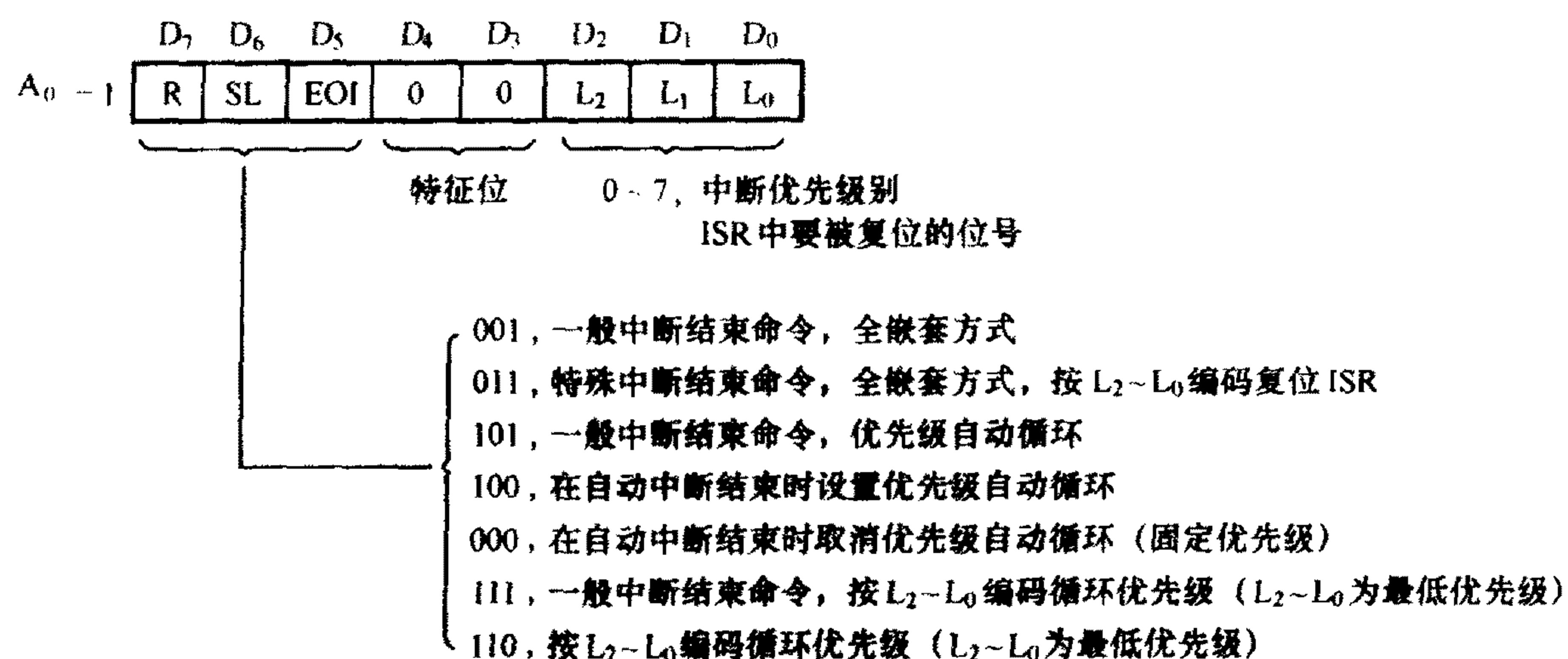


图 7.35 OCW₂ 操作命令字格式

3) OCW₃

OCW₃ 为屏蔽方式和状态读出控制字

OCW₃ 的格式如图 7.36 所示。

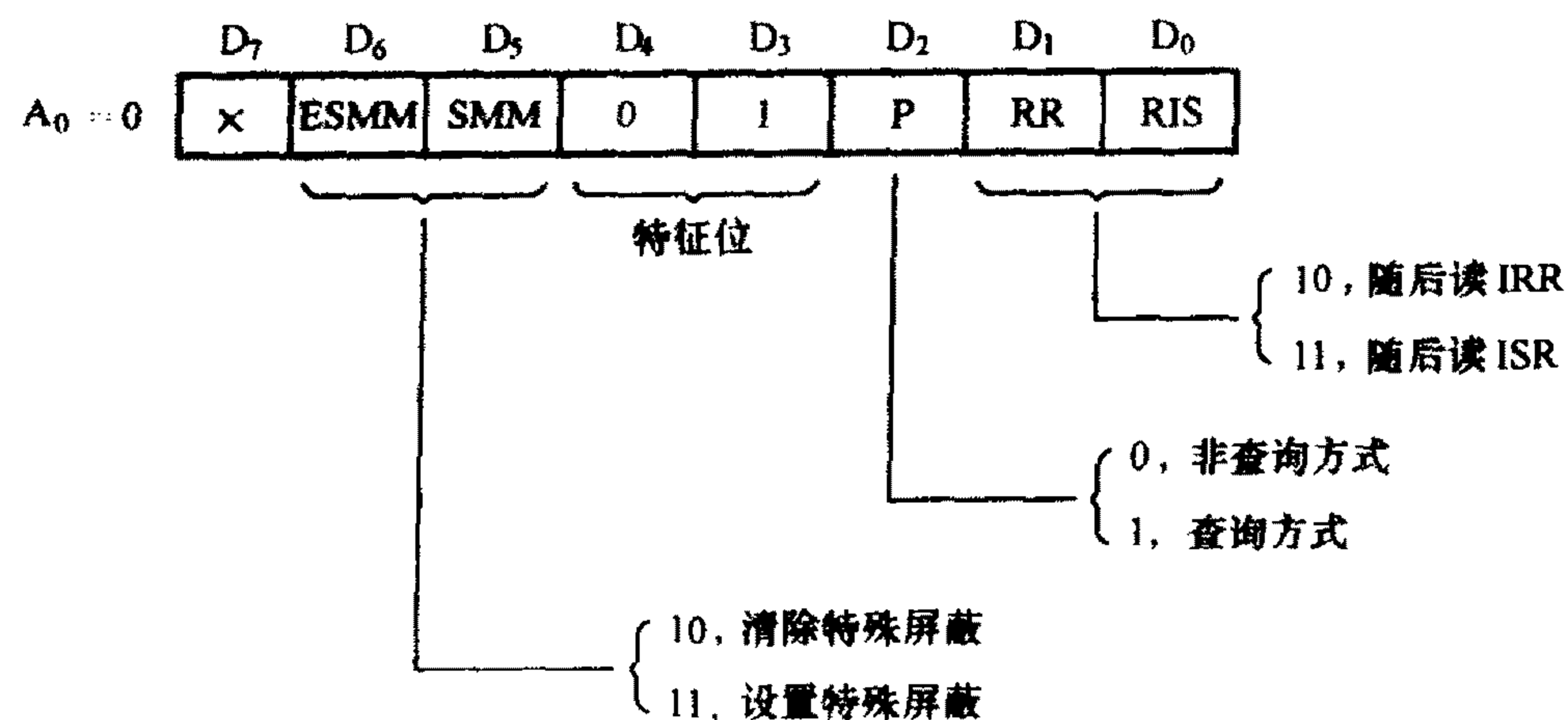


图 7.36 OCW₃ 操作命令字格式

该操作命令字具有以下 3 个功能：

① 设置中断屏蔽方式(见图中说明)。

② 查询中断请求 当 CPU 禁止中断或不希望 8259A 向 CPU 申请中断时,可以采用 8259A 的查询工作方式。CPU 先写一个 $P=1$ 的 OCW_3 到 8259A,再对同一地址读入,即可得到图 7.37 所示格式的状态字节。

此时,若 $I=1$,则表示本片 8259A 的 $IR_0 \sim IR_7$ 中有中断请求产生,其中最高优先级的 IR 线的编码由 $R_2 \sim R_0$ 给出; $I=0$,表示无中断请求产生(此查询步骤可反复执行,以响应多个同时发生的中断)。



图 7.37 8259A 中断状态查询结果

③ 读 8259A 状态 可用 OCW_3 命令控制读出 IRR、ISR 和 IMR 的内容。

CPU 先写一个 RR 和 RIS 位等于 10 的 OCW_3 到 8259A,再对同一地址读,即可读入 IRR 的内容。

CPU 先写一个 RR 和 RIS 位等于 11 的 OCW_3 到 8259A,再对同一地址读,即可读入 ISR 的内容。

而当 $A_0=1$ (奇地址)时读 8259A,则读出的都是 IMR 的内容(不依赖于 OCW_3)。

7.4.4 8259A 在微型计算机系统中的应用

1. 8259A 在 IBM PC/XT 微型计算机系统中的应用

IBM PC/XT 微型计算机系统 CPU 是 8086,系统只含一片 8259A 中断控制器,可处理 8 级外部可屏蔽中断(如图 7.38 所示),各级中断的类型号、向量地址指针、用途及其在系统 BIOS 中的过程名和首地址见表 7-1。其中,D11 程序不执行具体的操作,仅对是否是外

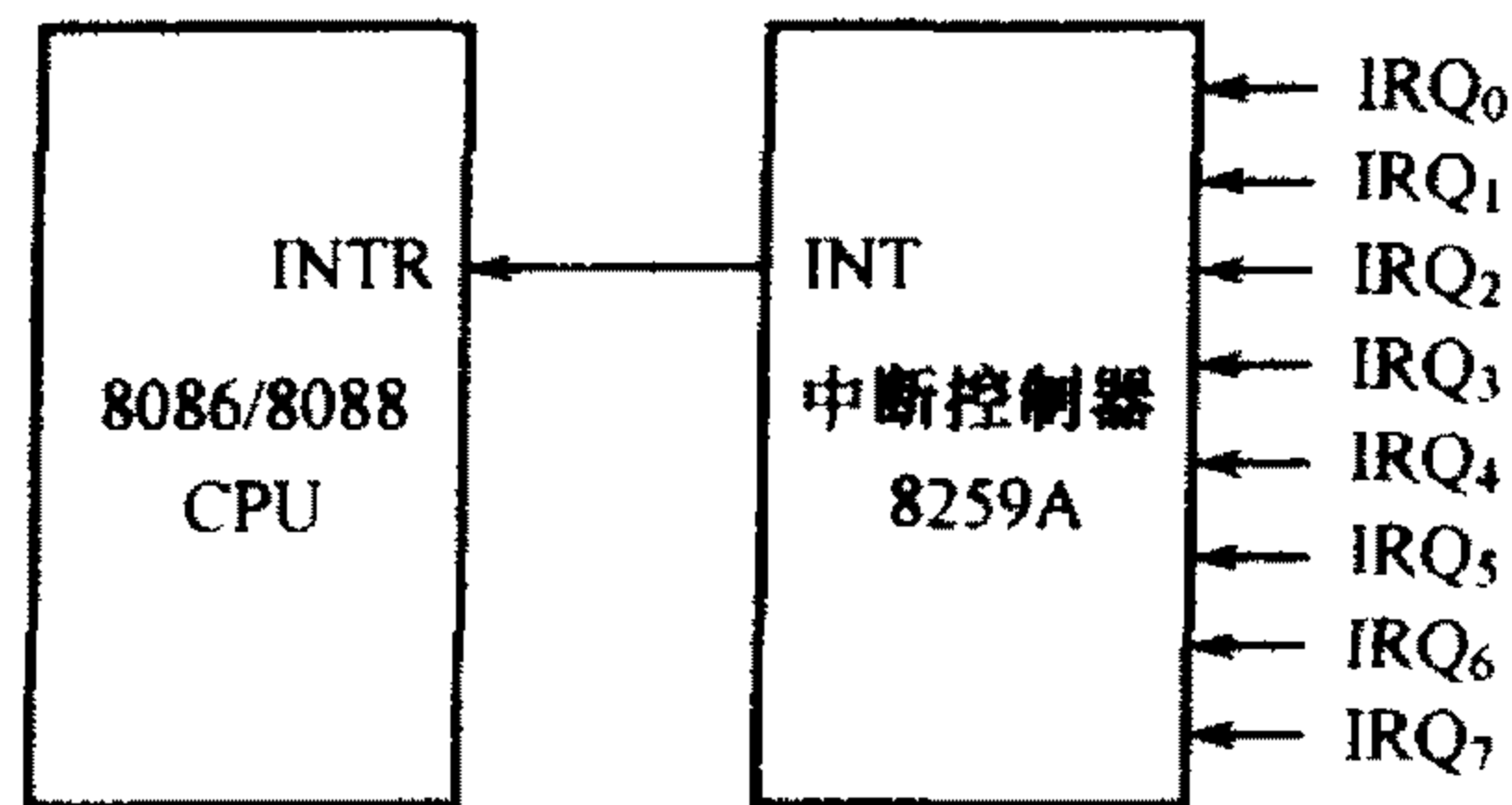


图 7.38 IBM PC/XT 机中的外中断设置

部中断做出判断,若是外部中断,就必须在中断服务程序中发 EOI 命令。

在 PC/XT 机外中断系统中,只有 IRQ_2 (即 IR_2) 端为保留端,可供用户使用,而 IRQ_3 、 IRQ_4 和 IRQ_7 则是在系统不用时(即未插接相应选件板,或没有使用中断)方可供用户使用。

表 7-1 IBM PC/XT 的 8 级外中断

中断向量地址指针	中断类型码	中断源	BIOS 中的中断服务程序过程名 (段地址: 偏移地址)
00020H	08H	日期、时钟	TIMER - INT(F000: FFA5H)
00024H	09H	键盘	KB - INT(F000: E987H)
00028H	0AH	保留	D11(F000: FF23H)
0002CH	0BH	串行口 2	D11(F000: FF23H)
00030H	0CH	串行口 1	D11(F000: FF23H)
00034H	0DH	硬盘	HD - INT(C800: 0760H)
00038H	0EH	软盘	DISK - INT(F000: EF57H)
0003CH	0FH	打印机	D11(F000: FF23H)

IBM PC/XT 微型计算机系统为 8259A 分配的 I/O 地址是 20H($A_0 = 0$, 偶地址)和 21H($A_0 = 1$, 奇地址)。BIOS 程序对芯片的初始化规定: 中断优先级管理采用一般全嵌套方式, 请求信号为上升沿触发、缓冲方式, 非自动中断结束。

;8259A 的初始化程序

```

                MOV AL,13H          ;写入 ICW1, 上升沿触发, 单片 8259A
                OUT 20H, AL
INTR1:          MOV AL,08H          ;写入 ICW2, 设定  $IRQ_0$  的中断类型码为 08H
                OUT 21H, AL
INTR3:          MOV AL,09H          ;写入 ICW4, 设定为一般全嵌套方式, 缓冲方式, 自动 EOI
                OUT 21H, AL
                :

```

若使用 IR_2 端作中断请求, 则中断类型码就要改变, 初始化程序中的 ICW_2 需重新写入。

2. 8259A 在 IBM PC/AT 微型计算机系统中的应用

IBM PC/AT 微型计算机中使用 80286 作微处理器。在 80286 以上的微型计算机系统中, 共使用了两片 8259A (新型的微型计算机已将中断控制器集成到了芯片组中, 但功能上与 8259A 完全兼容), 两片接成级联方式使用, 其连接如图 7.39 所示。从片的 INT 端接到主片的 IRQ_2 引脚, 相当于主片的 IRQ_2 又扩展了 8 个中断请求端 $IRQ_8 \sim IRQ_{15}$, 共可管理 15 级中断。主片的 INT 端直接与 CPU 的 INTR 端连接, 用于中断请求信号的输入。主从两片 8259A 的

CAS₀ ~ CAS₂ 信号互相连在一起,从主片输出到从片。从片中的 5 个保留端都可供用户使用。

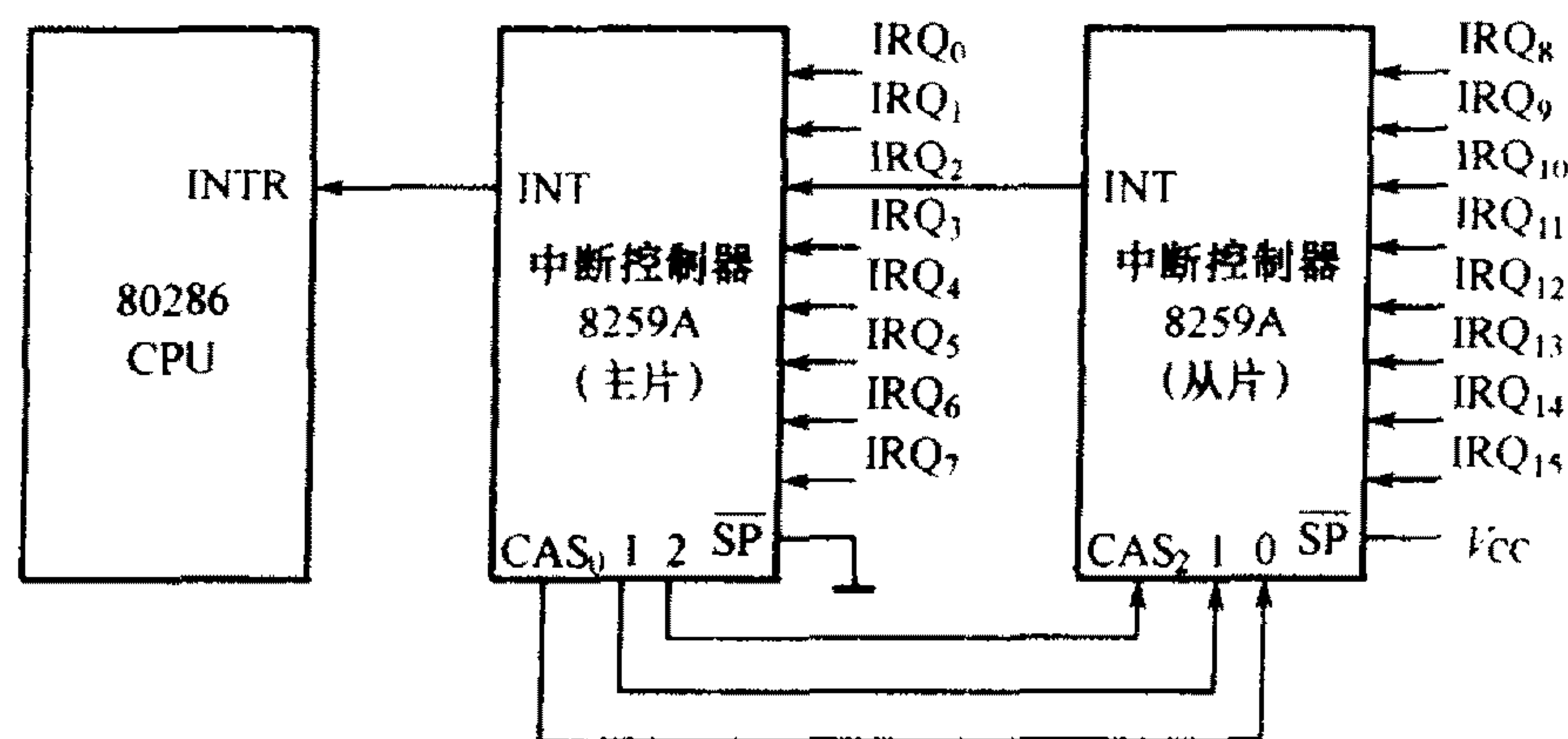


图 7.39 IBM PC/AT 机中的外中断设置

各级中断的用途见表 7-2。

表 7-2 IBM PC/AT 的中断源和类型号

中断向量地址指针	8259A 引脚	中断类型码	优先级	中断源
00020H	主片 IR ₀	08H	0(最高)	定时器
00024H	主片 IR ₁	09H	1	键盘
00028H	主片 IR ₂	0AH	2	从片 8259A
001C0H	从片 IR ₀	70H	3	时钟/日历钟
001C4H	从片 IR ₁	71H	4	IRQ ₉ (保留)
001C8H	从片 IR ₂	72H	5	IRQ ₁₀ (保留)
001CCH	从片 IR ₃	73H	6	IRQ ₁₁ (保留)
001D0H	从片 IR ₄	74H	7	IRQ ₁₂ (保留)
001D4H	从片 IR ₅	75H	8	协处理器
001D8H	从片 IR ₆	76H	9	硬盘控制器
001DCH	从片 IR ₆	77H	10	IRQ ₁₅ (保留)
0002CH	主片 IR ₃	0BH	11	异步通信口(COM2)
00030H	主片 IR ₄	0CH	12	异步通信口(COM1)
00034H	主片 IR ₅	0DH	13	并行打印口 2
00038H	主片 IR ₆	0EH	14	软盘驱动器
0003CH	主片 IR ₇	0FH	15(最低)	并行打印口 1

主片 8259A 的端口地址为 20H、21H, 中断类型码为 08H ~ 0FH, 从片 8259A 的端口地址为 A0H、A1H, 中断类型码为 70H ~ 77H。主片的 8 级中断已全被系统使用(其中 IRQ₂ 被从片

占用),从片尚保留 5 级未用。其中 IRQ_0 用于日历/时钟中断(08H), IRQ_1 用于键盘中断(09H)。扩展的 IRQ_8 用于实时时钟中断, IRQ_{13} 来自协处理器 80287。除上述中断请求信号外,所有其他的中断请求信号都来自 I/O 通道的扩展板。

;8259A 主片和从片的初始化程序

;主片 8259A 的初始化

```
MOV AL,11H          ;写入 ICW1,设定边沿触发,级联方式
OUT 20H,AL
JMP INTR1            ;延时,等待 8259A 操作结束,下同
INTR1: MOV AL,08H    ;写入 ICW2,设定  $IRQ_0$  的中断类型码为 08H
OUT 21H,AL
JMP INTR2
INTR2: MOV AL,04H    ;写入 ICW3,设定主片  $IRQ_2$  级联从片
OUT 21H,AL
JMP INTR3
INTR3: MOV AL,11H    ;写入 ICW4,设定特殊全嵌套方式,一般 EOI 方式
OUT 21H,AL
:
```

;从片 8259A 的初始化

```
MOV AL,11H          ;写入 ICW1,设定边沿触发,级联方式
OUT 0A0H,AL
JMP INTR5
INTR5: MOV AL,70H    ;写入 ICW2,设定从片  $IR_0$ ,即  $IRQ_8$  的中断类型码为 70H
OUT 0A1H,AL
JMP INTR6
INTR6: MOV AL,02H    ;写入 ICW3,设定从片级联到主片的  $IRQ_2$ 
OUT 0A1H,AL
JMP INTR7
INTR7: MOV AL,01H    ;写入 ICW4,设定普通全嵌套方式,一般 EOI 方式
OUT 0A1H,AL
:
```

当来自某个从片的中断请求进入服务时,主片的优先级控制逻辑不封锁这个从片,从而使来自从片的更高优先级的中断请求能被主片所识别,并向 CPU 发出中断请求信号。因此,中断服务程序结束时必须用软件来检查被服务的中断是否是该从片中惟一的中断请求。先向从片发出一个 EOI 命令,清除已完成服务的 ISR 位,然后再读出 ISR 的内容,检查它是否为 0。如果 ISR 的内容为 0,则向主片发一个 EOI 命令,清除与从片相对应的 ISR 位;否则,

就不向主片发 EOI 命令,继续进行从片的中断处理,直到 ISR 的内容为 0,再向主片发出 EOI 命令。级联工作时的程序段如下:

```
      ;读 ISR 的内容
      MOV AL,0BH          ;写入 OCW3,读 ISR 命令
      OUT 0A0H,AL
      NOP                  ;延时,等待 8259A 操作结束
      IN AL,0A0H          ;读出 ISR
      :
      ;向从片发 EOI 命令
      MOV AL,20H
      OUT 0A0H,AL         ;写从片 EOI 命令
      :
      ;向主片发 EOI 命令
      MOV AL,20H
      OUT 20H,AL          ;写主片 EOI 命令
      :
```

习 题 七

- 7.1 输入/输出系统具有哪些特点和基本功能?
- 7.2 I/O 接口的基本功能有哪些?
- 7.3 什么是 I/O 端口? 端口地址有哪两种编址方式? 在 8088/8086 系统中采用哪一种编址方式?
- 7.4 计算机系统中常用的输入/输出方法有哪些? 它们各自具有哪些特点?
- 7.5 微型计算机系统常用的三种基本输入/输出方法中,哪一种方式下 CPU 的效率最高?
- 7.6 什么是中断? 在何种情况下中断可以被屏蔽?
- 7.7 CPU 满足什么条件能够响应可屏蔽中断?
- 7.8 与 DMA 传送方式相比,中断控制方式有什么不足?
- 7.9 试说明在 DMA 方式时由内存向外设传送数据的过程。
- 7.10 在响应中断时,8086 在 INTA 端送出连续的两个负脉冲,它们起什么作用?
- 7.11 8086 最多可管理多少级中断? 按中断源的位置不同,中断可分为哪两大类?
- 7.12 简述中断响应的 5 个步骤。
- 7.13 在 8086 系统中,中断向量表处于内存的哪个区域中?
- 7.14 若已知一个中断源的中断向量码为 48H,则其中断服务程序的入口地址存放在内存的哪 4 个单元中?

-
- 7.15 8259A 的一般全嵌套方式和特殊全嵌套方式有什么差别？各自用于什么场合？
- 7.16 试编写 8259A 的初始化程序：系统中仅有一片 8259，允许 8 个中断源边沿触发，不需要缓冲，一般全嵌套方式工作，中断向量为 48H。
- 7.17 单片 8259A 能够管理多少级可屏蔽中断？若用 6 片级联，能管理多少级可屏蔽中断？
- 7.18 试将中断向量码为 40H 的中断服务程序的入口地址放入中断向量表中。
- 7.19 在保护模式下，中断服务程序的入口地址由什么提供？
- 7.20 中断描述符表由 8 字节的描述符组成，它们分别是什么？

第8章 输入/输出接口

接口是 CPU 与外部设备之间进行信息交换的必经通道,这种“角色”,决定了接口需要完成信息缓冲、信息转换、电平转换、数据存取和传送、联络控制等工作,这些工作分别由接口电路的两大部分即与计算机连接的总线接口和与外部设备连接的外设接口来实现。总线接口一般包括内部寄存器、存取逻辑和传送控制逻辑电路等,主要负责数据缓冲、传输管理等工作;而外设接口则负责与外部设备通信时的联络和控制以及电平和信息转换等。总线接口在第3章中已做过介绍,以下如无特殊说明,所讨论的接口电路均指外设接口。

接口电路从总的功能上可以分为输入接口和输出接口,分别完成信息的输入和输出。从传送方式上,又可分为并行接口和串行接口。另外,从所传送信息的类型上,还可分为数字量的输入/输出(I/O)接口及模拟量的输入/输出接口。

近年来,随着超大规模集成电路技术的发展,I/O 接口电路的集成度和智能化程度越来越高,单个接口集成电路芯片的集成度通常超过几万个元器件/片。集成度的提高使得在接口芯片内直接集成 I/O 专用微处理器成为可能,这就是所谓的智能化的 I/O 接口。智能 I/O 接口能够根据主机发送的命令执行相应的控制程序,完成非常复杂的 I/O 操作,这样就大大减轻了 CPU 的负担。I/O 接口的大规模集成化也降低了外围电路的复杂性,从而提高了电路的可靠性。

本章以某些典型的接口芯片为例,分数字量 I/O 接口及模拟量 I/O 接口两个部分,来介绍接口电路的构成及其具体应用方法。

8.1 简单数字接口电路

8.1.1 接口电路的基本构成

负责把信息从外部设备送入 CPU 的接口称为输入接口,而把信息输出到外部设备的接口则称为输出接口。

在需要从外设输入数据时,由于外设的速度相对于 CPU 要慢得多,这意味着数据在外部总线上保持的时间相对较长,所以要求输入接口必须要具有对数据的控制能力,即要在外部数据准备好,CPU 可以读时才允许将数据送上系统数据总线。大多数外设都具有数据保

持能力(即 CPU 没有读取时,外设能够保持数据不变),通常可以仅用一个三态门缓冲器作为输入接口。当三态门的控制端信号有效时,三态门导通,该外设就与数据总线连通,CPU 将外设准备好的数据读入;当控制端信号无效时,三态门断开,该外设就从数据总线脱离,数据总线又可用于其他信息的传送。对没有数据保持能力的外设,可在外设与接口之间增加一个锁存器,用外设提供的数据准备好信号把数据保存到锁存器中。

在数据输出时,同样应考虑外设与 CPU 速度的配合问题。要使数据能正确写入外设,CPU 输出的数据一定要能够保持一段时间。但从前面介绍的总线写时序图可以看出,CPU 送到总线上的数据只能保持几个微秒甚至更短的时间。相对于慢速的外设,数据在总线上几乎是一闪而逝。因此,要求输出接口必须要具有数据的锁存能力,这通常是由锁存器来实现的。CPU 输出的数据通过总线锁存到锁存器中,并一直保持到被外设取走。

以上三态门和锁存器的控制端一般与 I/O 地址译码输出信号线相连,当 CPU 执行 I/O 指令时,指令中指定的 I/O 地址经译码后即可使控制信号有效,打开三态门(对外设读时)或将数据锁入锁存器(对外设写时)。

8.1.2 基本输入接口

三态缓冲器(也叫三态门)常用来构造输入接口。一个典型的三态缓冲器芯片 74LS244 如图 8.1 所示。从图中不难看出该芯片由 8 个三态门构成。74LS244 有两个控制端: \bar{E}_1 和 \bar{E}_2 ,每个控制端各控制 4 个三态门。当某一控制端有效(低电平)时,相应的 4 个三态门导通;否则,相应的三态门呈现高阻状态(断开)。实际使用中,通常是将两个控制端并联,这样就可用一个控制信号来使 8 个三态门同时导通或同时断开^①。

由于三态门具有控制信号通过的能力,故可利用其作输入接口。利用三态门作为输入接口时,要求外设数据信号的状态能够保持到 CPU 完全读入为止,这是因为三态门本身没有对信号的保持或锁存能力。图 8.2 所示是一个利用 74LS244 作为开关量输入接口的例子。图中,74LS244 的输入端接有 8 个开关 S_1, S_2, \dots, S_8 ,其输出端接到数据总线上。当 CPU 读该接口(对 80X86 系列 CPU 来说,就是执行 IN 指令)时,总线上的 16 位地址信号通过译码使 \bar{E}_1 和 \bar{E}_2 有效(逻辑 0),于是三态门导通,8 个开关的状态经数据线 $D_0 \sim D_7$ 被读

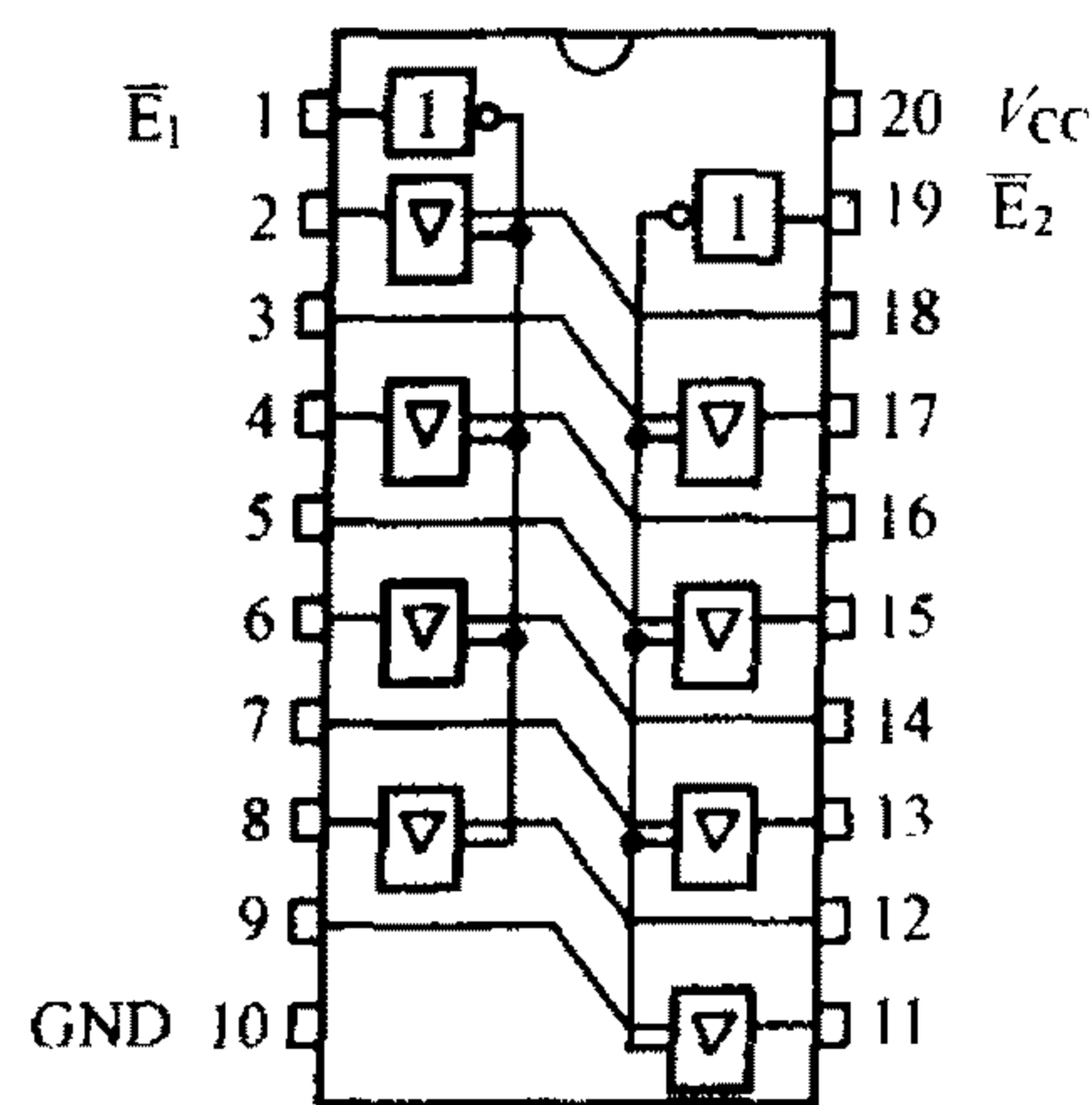


图 8.1 74LS244 芯片引出线图

① 大多数 I/O 操作每次同时传送 8 位数据。

入到 CPU 中。这样,就可测量出这些开关当前的状态是打开还是闭合。当 CPU 不访问此接口地址时, \bar{E}_1 和 \bar{E}_2 为高电平(逻辑 1),则三态门的输出为高阻状态,使其与数据总线断开。

用一片 74LS244 芯片作为输入接口,最多可以连接 8 个开关或其他具有信号保持能力的外设。当然也可只接一个外设而让其他端悬空,对空着未用的引出线,其对应位的数据是任意值,在程序中常用逻辑与指令将其屏蔽掉。如果有更多的开关状态(或其他外设)需要输入时,可用类似的方法用两片或更多的芯片并联使用。

假如要求编写程序段判断图中的开关的状态。若所有的开关都闭合,则程序转向标号为 NEXT1 的程序段执行,否则转向标号为 NEXT2 的程序段执行。

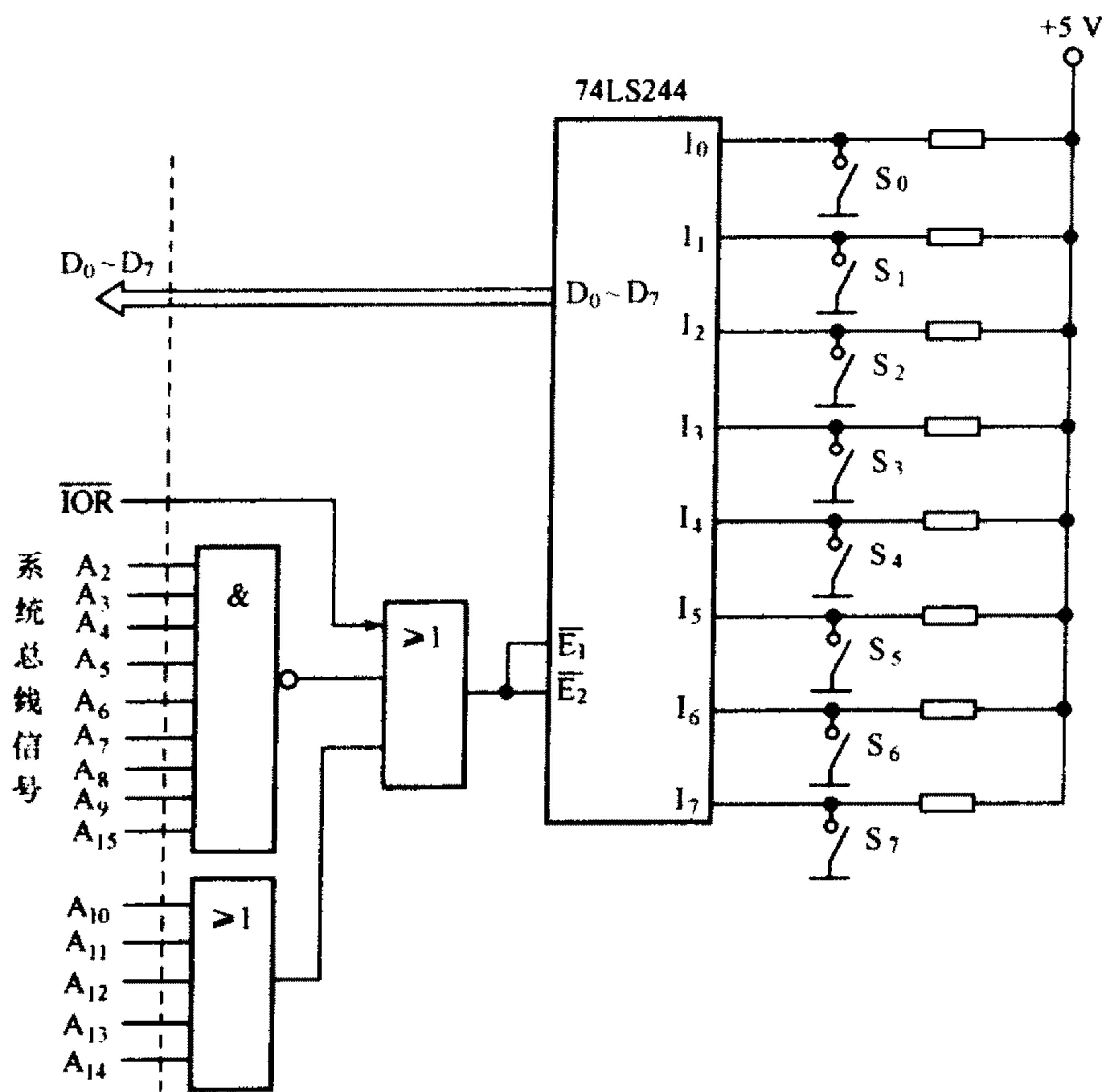


图 8.2 三态缓冲器构成的输入接口

图 8.2 中,三态缓冲器 74LS244 的 I/O 地址采用了部分地址译码——地址线 A_1 和 A_0 未参加译码,故它占用了 83FCH ~ 83FFH 之间共 4 个地址。其中任何一个地址都可以选中 74LS244 缓冲器。另外,由图可以看出,当某个开关闭合时,其对应的输入为低电平(逻辑 0),当某个开关打开时,其对应的输入为高电平(逻辑 1)。读开关状态的程序段如下:

```
MOV  DX,83FCH
IN   AL,DX
```



```
AND    AL,0FFH
JZ      NEXT1
JMP     NEXT2
...
```

8.1.2 基本输出接口

由于三态门不具备保存(锁存)数据的能力,因此它不能直接用做数据输出接口。数据输出接口通常是用具有信息存储能力的双稳态触发器来实现。最简单的输出接口可用 D 触发器构成。例如,常用的锁存器 74LS273,它内部包含了 8 个 D 触发器。其引出线排列图及真值表如图 8.3 所示。

74LS273 共有 8 个数据输入端($D_0 \sim D_7$)和 8 个数据输出端($Q_0 \sim Q_7$)。 S 为复位端,低电平有效。 CP 为脉冲输入端,在每个脉冲的上升沿将输入端 D_i 的状态锁存在 Q_i 输出端,并将此状态保持到下一个时钟脉冲的上升沿。74LS273 常用来作为并行输出接口。另外,使用其中的某一个 D 触发器也可通过软件编程实现简单的串行输出。图 8.4 所示的是应用 74LS273 作为输出接口的例子。8 个 Q 端与 8 个发光二极管相连接,8 个数据端接到数据总线上,地址译码电路的输出接 CP 端。要使接到 Q_0 端和 Q_6 端的发光二极管发光,其对应的

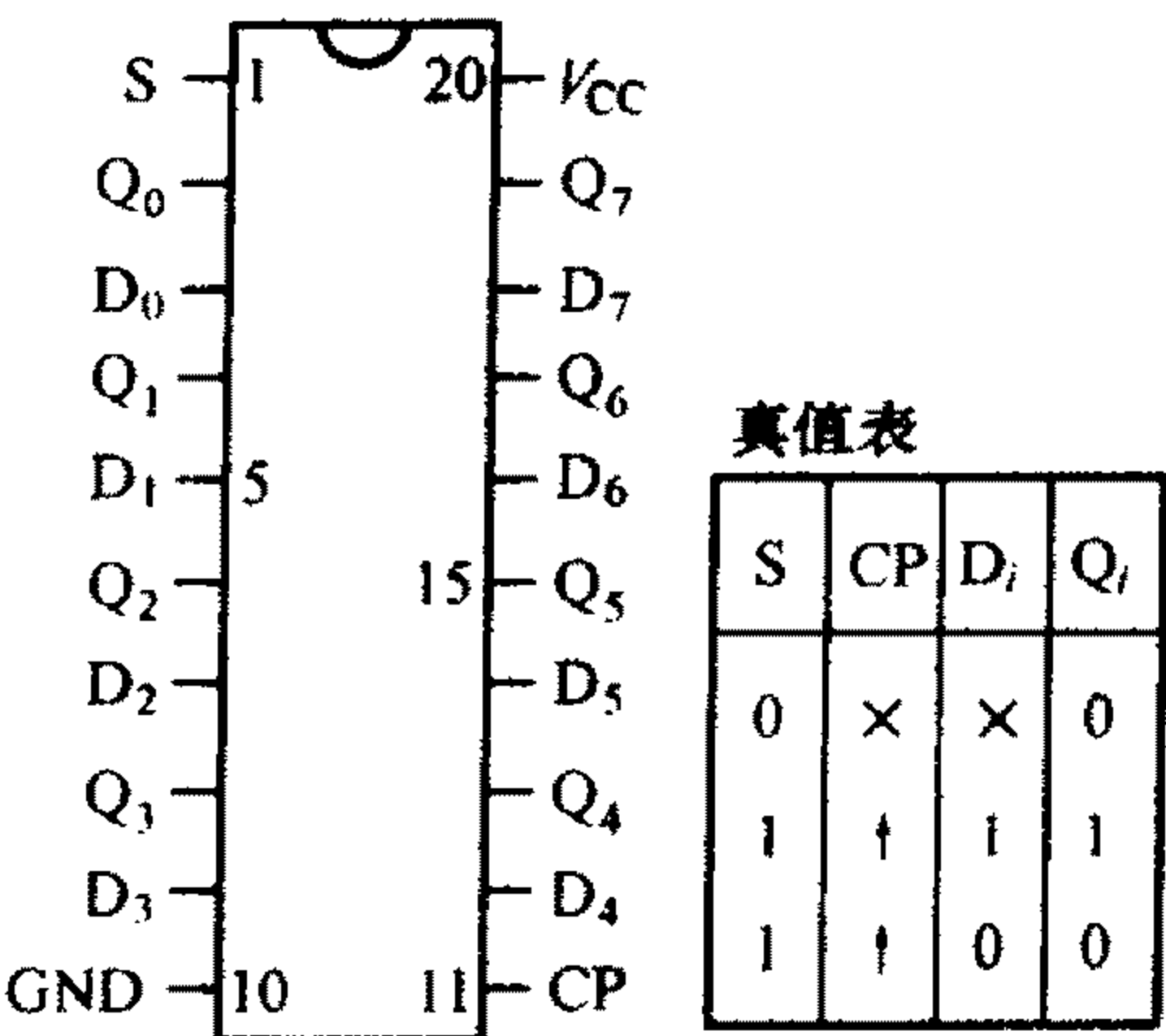


图 8.3 74LS273 引出线排列图和真值表

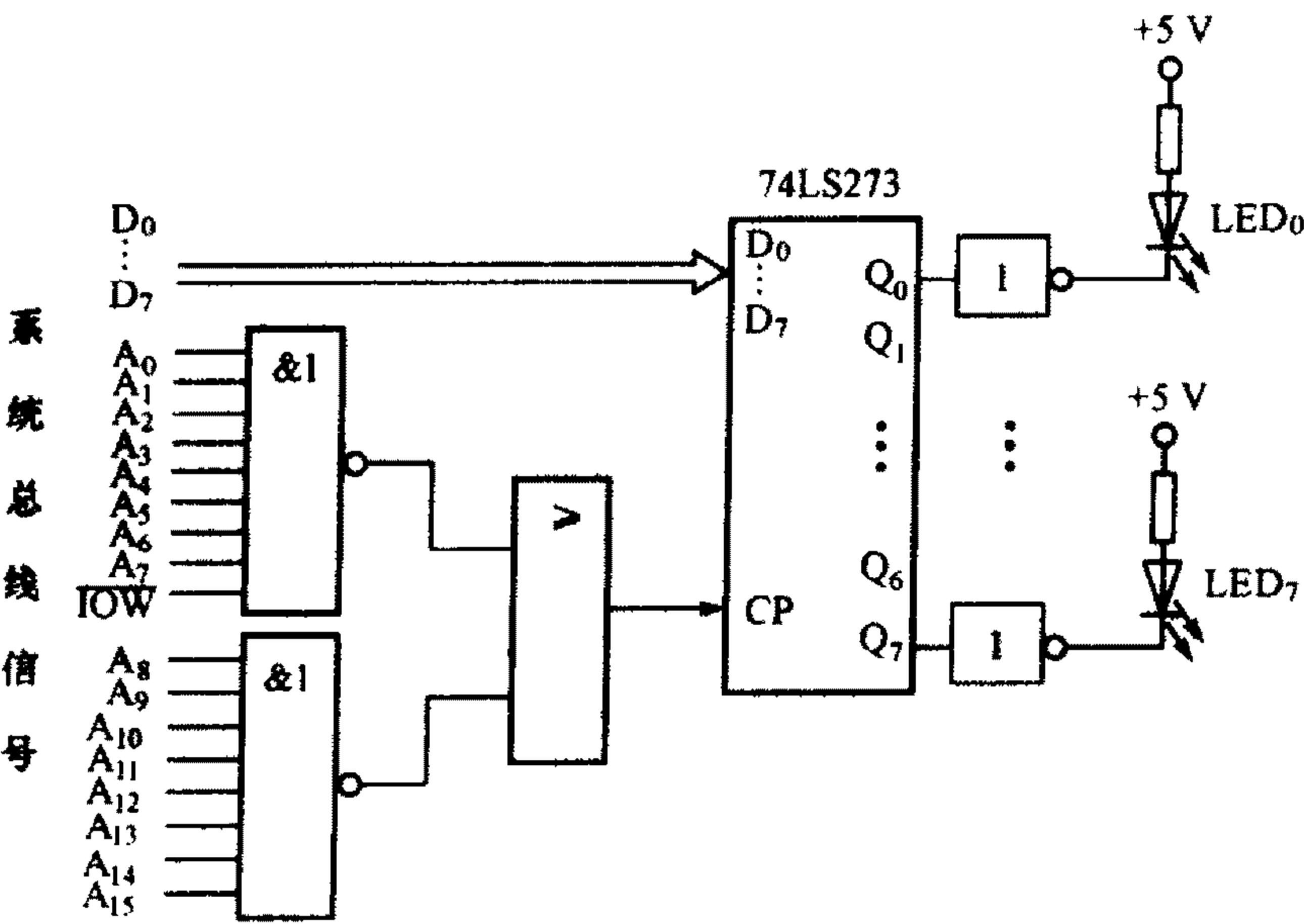


图 8.4 74LS273 用做 LED 显示器的输出接口

Q_0 、 Q_6 端需为“1”状态,而其他 Q 端则为“0”状态。假定该输出接口的地址为 0FFFFH,则使 N_0 和 N_6 发光的程序段如下:

```
MOV DX,0FFFFH
MOV AL,01000001B
OUT DX,AL
```

8.1.3 具有三态输出的锁存器

74LS273 的数据输出端 Q_i 不是三态输出的。也就是说,只要 74LS273 正常工作,其 Q 端总有一个确定的逻辑状态(0 或 1)输出。因此,74LS273 无法直接用做输入接口,即它的 Q_i 端绝对不允许直接与系统的数据总线相连接。那么,有没有既可做输入接口又可做输出接口的芯片呢? 回答是肯定的。这里介绍一种带有三态输出的锁存器 74LS374。这也是经常用到的一种电路芯片,其引出线排列图和真值表如图 8.5 所示。从引出线上可以看出,它比 74LS273 多了一个输出允许端 \overline{OE} 。只有当 $\overline{OE} = 0$ 时,74LS374 的输出三态门才导通; $\overline{OE} = 1$ 时,则呈高阻状态。图 8.6 所示为 74LS374 中一个锁存器的结构图。可以看出,74LS374 在 D 触发器输出端加有一个三态门。

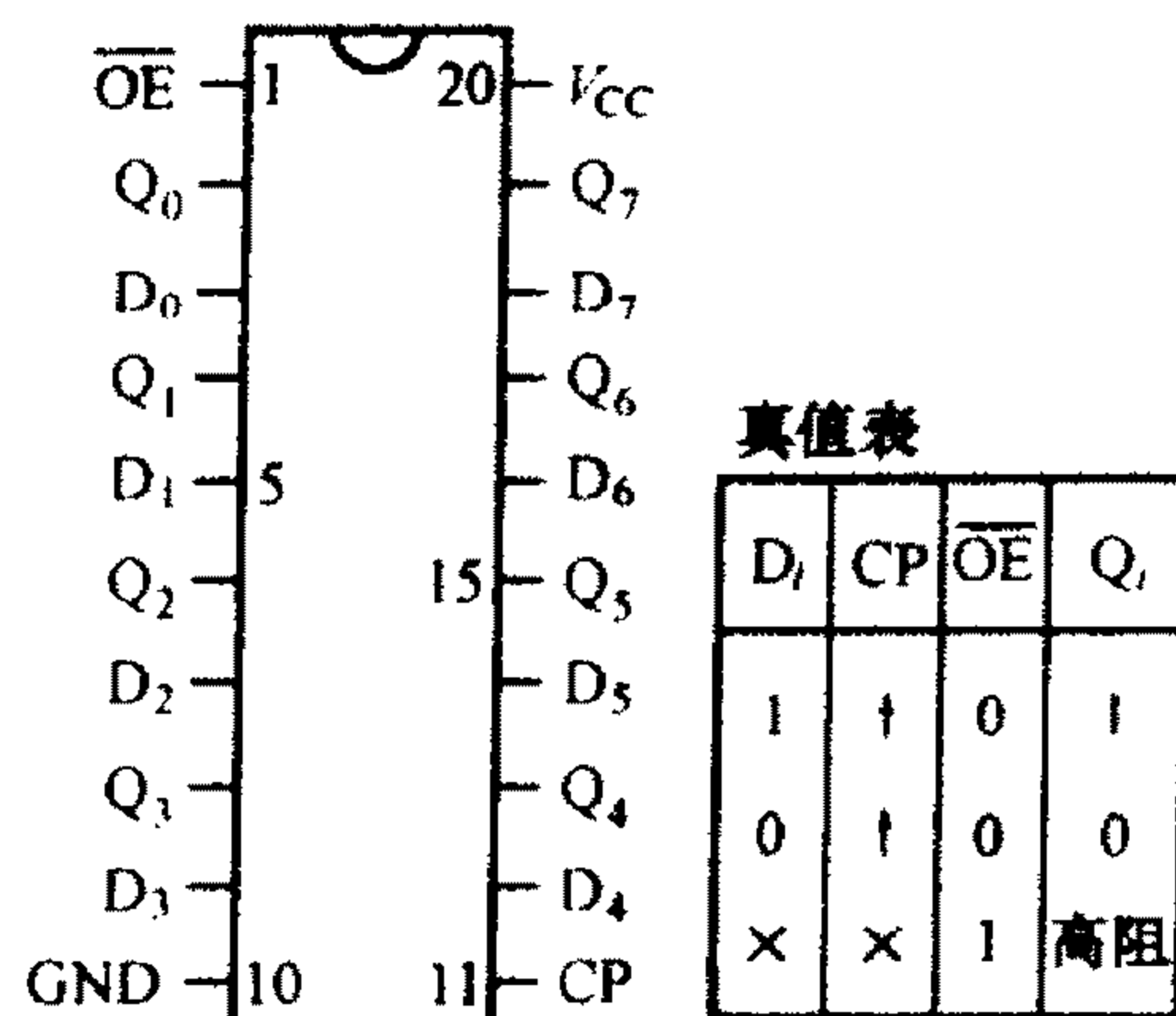


图 8.5 74LS374 引出线排列图和真值表

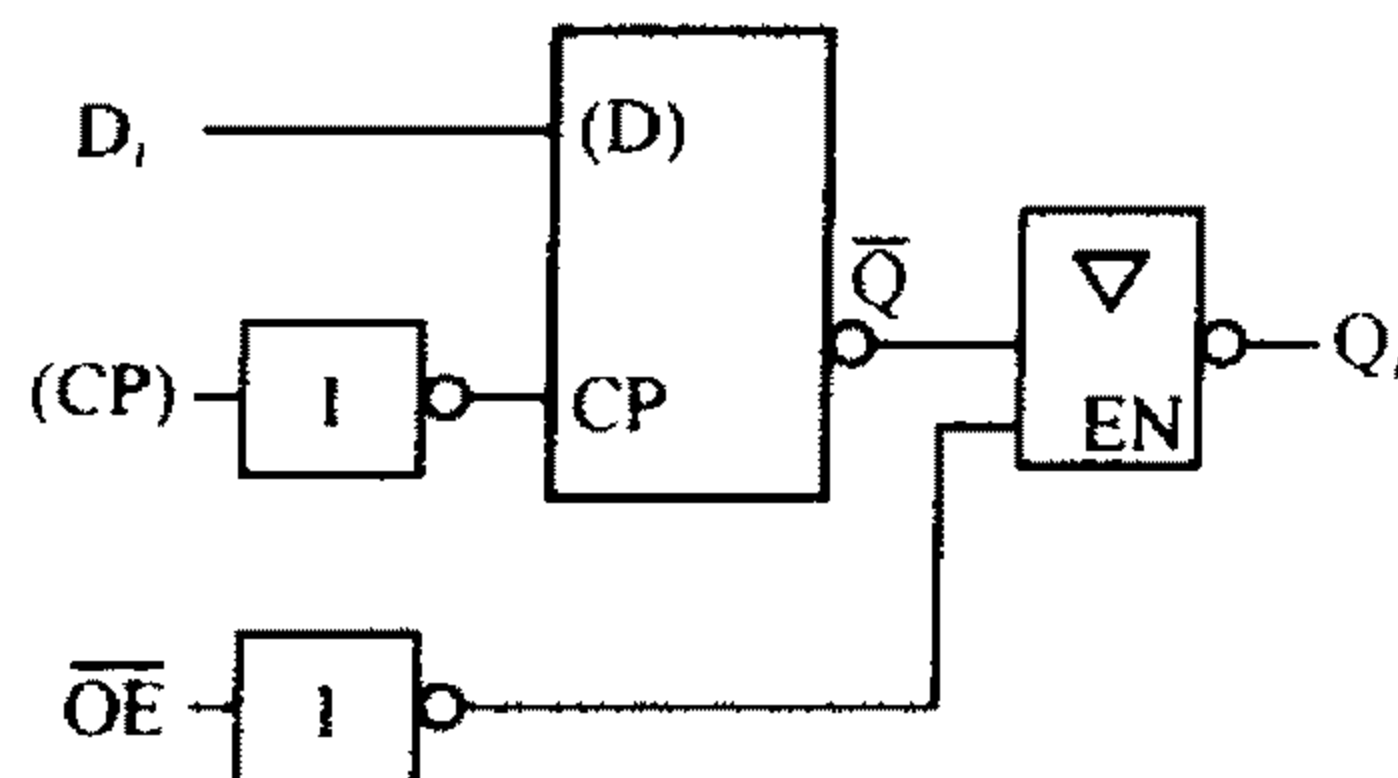


图 8.6 74LS374 内部结构

74LS374 在用做输入接口时,CPU 提供的端口地址信号经译码电路接到 \overline{OE} 端,数据由外设提供的选通脉冲(接到 74LS374 的 CP 端)锁存在 74LS374 内部。当 CPU 读该接口时,译码器输出低电平,使 74LS374 的输出三态门打开,数据送到总线上由 CPU 读入。如果把 74LS374 用做输出接口,可将 \overline{OE} 端接地,使其输出三态门一直处于导通状态,这样就与 74LS273 一样使用了。

用 74LS374 分别作为输入接口和输出接口的电路如图 8.7 所示。

另外还有一种常用的带有三态门的锁存器芯片 74LS373,它与 74LS374 在结构和功能上

完全一样,区别是数据锁存的时机不同,74LS373 是在 CP 脉冲的高电平期间将数据锁存。

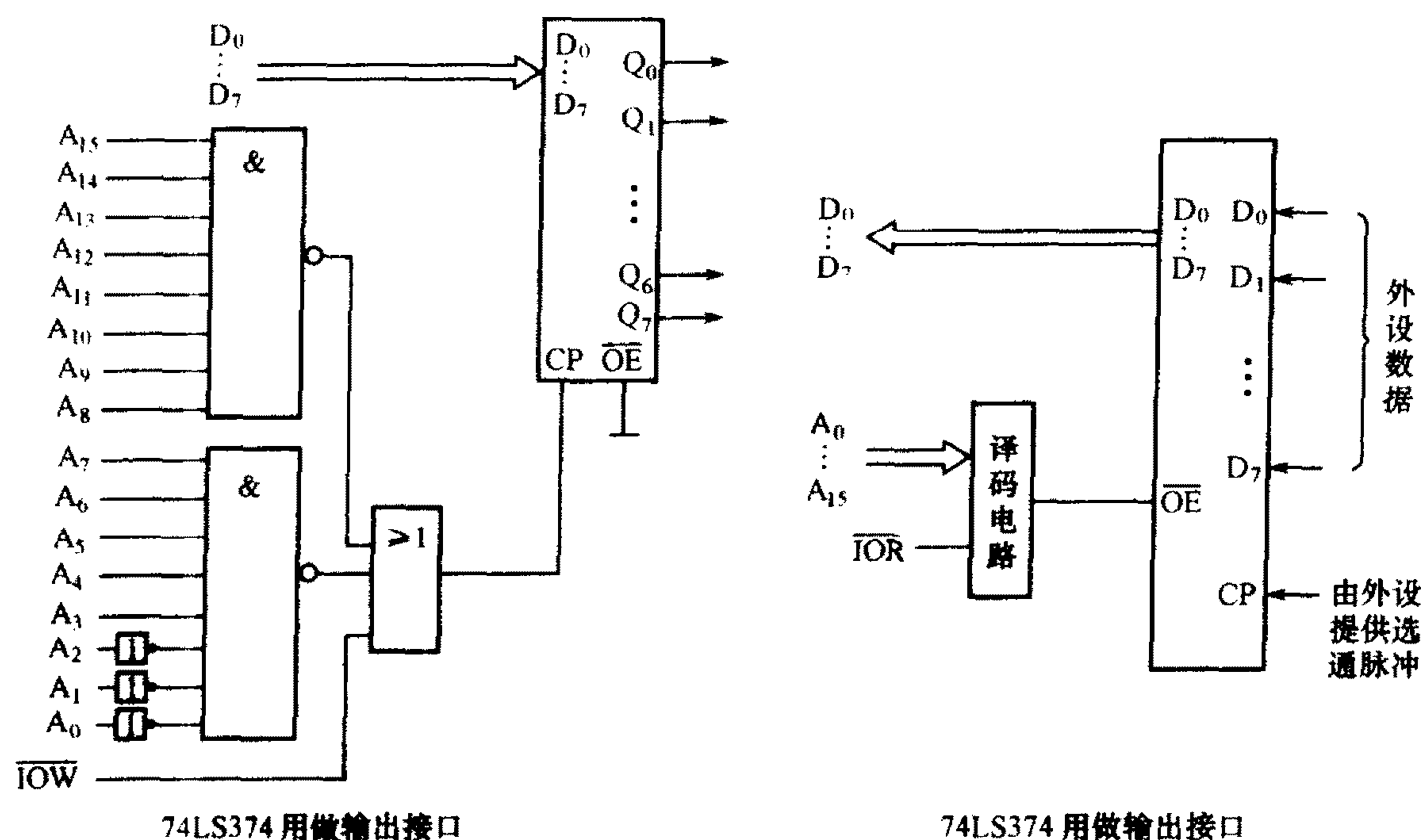


图 8.7 74LS374 用做输入和输出接口

从以上的介绍可以看出,三态缓冲器和锁存器在构造上比较简单,使用也很方便。常作为一些功能简单的外部设备的接口电路。但由于它们的功能有限,对较复杂的功能要求就难以胜任。

8.1.4 简单接口的应用举例

在本小节中,主要介绍如何利用 74LS244 和 74LS273 作为输入和输出接口,通过编写程序,控制 LED 数码管显示不同的数字或符号。

1. LED 数码管

LED 数码管分为共阳极和共阴极两种结构。在封装上有将 1 位、2 位或更多位封装在一起的。由于篇幅限制,这里只介绍一种共阳极封装的 LED 数码管,如图 8.8 所示。当某一段的发光二极管流过一定电流(例如 10 mA 左右)时,它所对应的段就发光,而无电流流过时,则不发光。不同发光段的组合就可显示出不同的数字和符号。表 8-1 列出了一些常用符号的七段码。

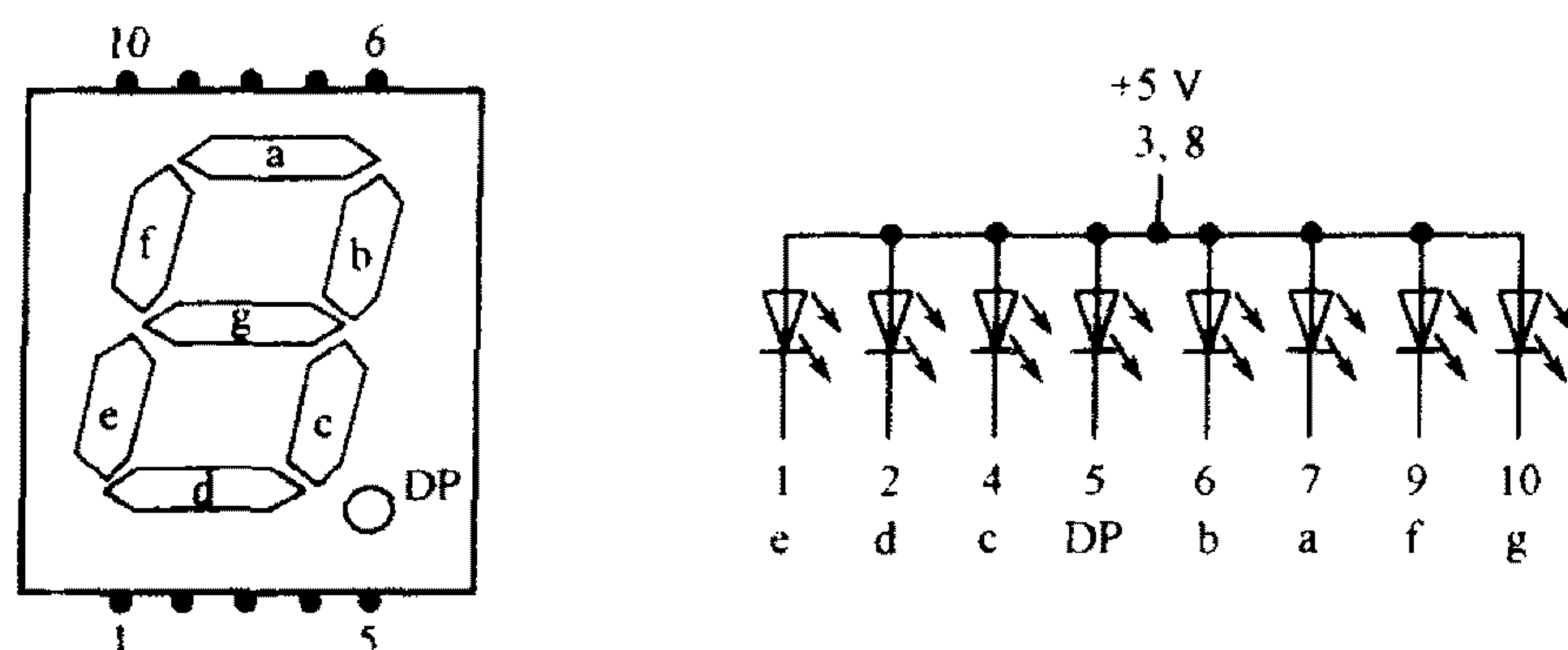


图 8.8 共阳极 LED 数码管示意图

表 8-1 常用符号的七段码表

符号	形状	七段码 gfedcba	符号	形状	七段码 gfedcba
0	0	0111111	8	8	0111111
1	1	0000110	9	9	0110111
2	2	1011011	A	A	0111011
3	3	1001111	B	B	0111100
4	4	1100110	C	C	00111001
5	5	1101101	D	D	01011110
6	6	1111101	E	E	01111001
7	7	0000111	F	F	01110001

2. 应用与连接

七段数码管作为一种外设与系统总线有多种接口方式,这里利用前面介绍的 74LS273 作为输出接口,其输出端用集电极开路门 74LS06 来驱动 LED 数码管。74LS273 的地址为 0F0H。输入接口采用 74LS244,其地址为 0F1H。实际电路连接如图 8.9 所示。

图中的电路功能是:当开关 S 处于闭合状态时,在 LED 数码管上显示“0”,当开关 S 处于断开状态时,在 LED 数码管上显示“1”。与硬件电路相配合完成此功能的程序段如下:

```

FOREVER: MOV    DX,0F1H      ;输入端口地址为 0F1H
           IN     AL,DX       ;读入开关状态
           TEST   AL,1        ;判断开关状态

```

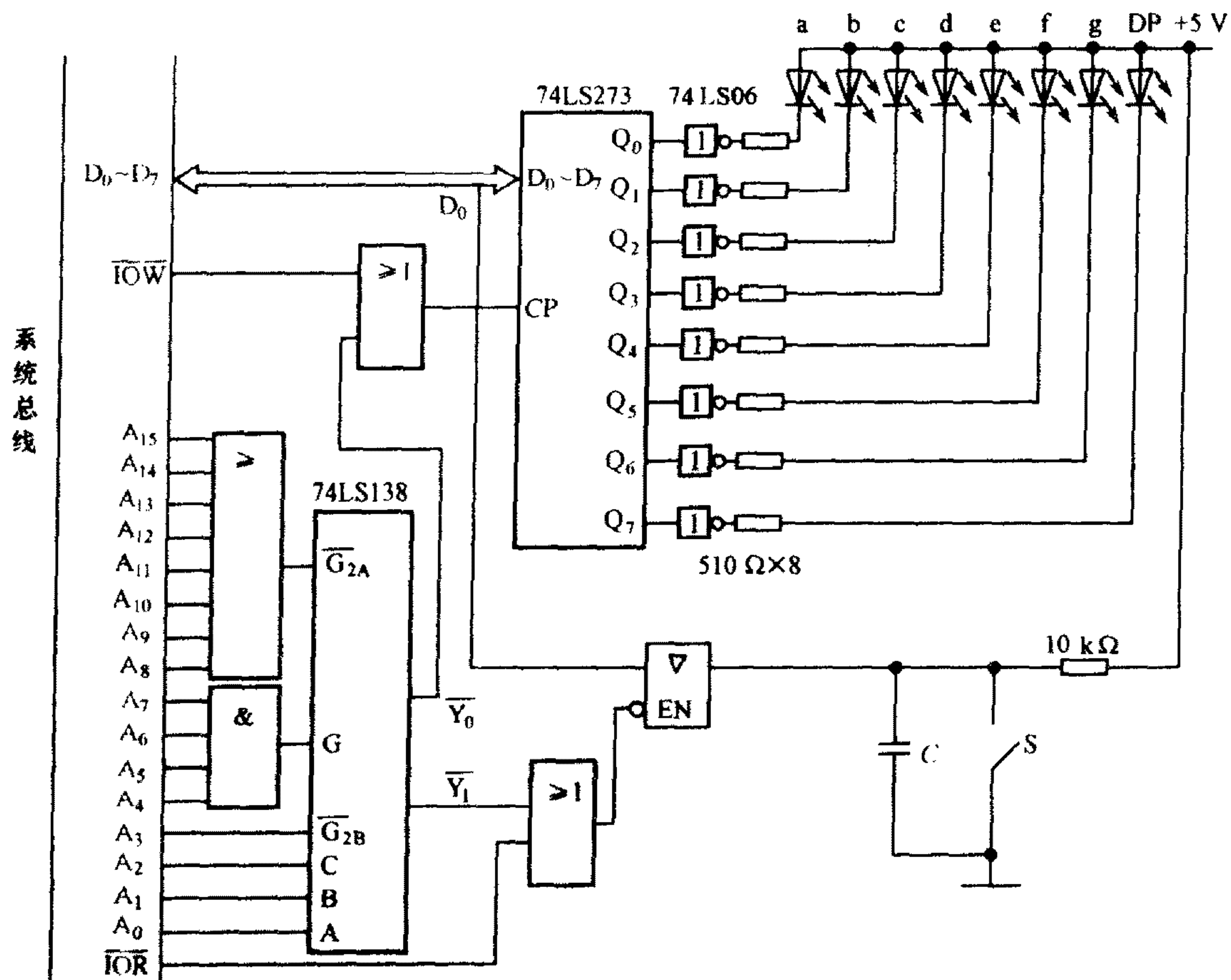



图 8.9 简单接口中的应用

```

MOV    AL,3FH          ;如果开关闭合,就显示“0”
JZ     DISP
MOV    AL,06H          ;否则,显示“1”
DISP:  MOV    DX,0F0H    ;输出端口地址为 0F0H
        OUT    DX,AL     ;显示
        JMP    FOREVER

```

8.2 可编程数字接口芯片

简单的接口电路一般只适合于慢速且功能比较简单的外设,难以满足各种复杂应用控制系统的要求。而要实现复杂控制应用的要求,接口电路就变得非常复杂,导致了应用系统的开发难度高,同时带来了开发周期长、效率低、可靠性低等弊病。为了解决这些问题,许多 IC 厂商开发了各种通用的 I/O 接口电路,把很复杂的电路集成到一片或几片大规模集成电

路芯片中。这样就使得系统开发人员能够快速容易地使用这些芯片构成所需功能的 I/O 接口。为了使通用性更强,这类芯片通常可以用命令来设置其工作模式,以满足不同的控制要求。这种设置工作模式的操作通常被称为对该芯片“编程”。而这类芯片也被称为可编程接口芯片。下面介绍几种常用的可编程的数字量 I/O 接口芯片和它们的应用。

8.2.1 可编程定时/计数器 8253

在数字电路、计算机系统以及实时控制系统中常常需要用到定时信号,如函数发生器、计算机中的系统日历时钟、DRAM 的定时刷新和实时采样等。产生这些定时信号既可以用软件编程的方法,也可以用硬件的方法得到。

所谓软件定时的方法就是设计一个延时子程序,子程序中全部指令执行时间的总和就是该子程序的延时时间。在 CPU 时钟频率一定时,子程序的延时时间是固定的。这种方法比较简单,较易实现,只是需要了解延时子程序中每条指令的执行时间。软件定时的定时时间不太精确,但使用方便,因此在软件开发中经常用到。这种方法仅适用于延时时间较短、重复次数有限的场合,否则,CPU 总是执行延时程序,占用了大量的时间,使 CPU 的利用率降低。故在对时间要求严格的实时控制系统和多任务系统中很少采用。

硬件的方法是利用专用的硬件定时/计数器,在简单软件控制下产生准确的延时时间。其基本原理是通过软件确定定时/计数器的工作方式、设置计数初值并启动计数器工作,当计数到给定值时,便自动产生定时信号。这种方法的成本不高,程序上也很简单,且几乎不占用 CPU 资源,既适合长时间、多次重复的定时,也可用于延时时间较短的场合,因此得到了广泛的应用。

另外,在控制系统中还经常要对外界的某种事件进行计数,如统计传送带上的产品、工件的数量等。对这种需求,也可以利用专用的硬件定时/计数器来实现。

定时/计数器在计数方式上分为加法计数器和减法计数器。加法计数器是每有一个计数脉冲就加 1,当加到预先设定的计数值时,产生一个定时信号;减法计数器是在送入计数初值后,每来一个计数脉冲就减 1,减到 0 时产生一个定时信号输出。以下要介绍的可编程定时/计数器 8253 就是一个减法计数器,它是 Intel 公司专为 80X86 系列 CPU 配套使用的 16 位可编程定时/计数器芯片。

1. 8253 的外部引脚及内部结构

1) 引脚及功能

8253 是一个三通道的 16 位定时/计数器芯片,每个计数器的工作方式及计数常数分别由软件编程选择,可进行二进制或十进制计数或定时操作,最高计数频率为 2 MHz。使用单电源 +5 V 供电,输入/输出均与 TTL 电平兼容。8253 是一个 24 根引脚的双列直插式器件,

其外部引线如图 8.10 所示。

8253 与系统总线连接的引脚包括：

$D_0 \sim D_7$ 8 位双向数据线。用来传送数据、控制字和计数器的计数初值。

\overline{CS} 选片信号,输入,低电平有效。由系统高位 I/O 地址译码产生。当它有效时,此定时/计数器芯片被选中。

\overline{RD} 读控制信号,输入,低电平有效。当它有效时,表示 CPU 要对此定时/计数器芯片进行读操作。

\overline{WR} 写控制信号,输入,低电平有效。当它有效时,表示 CPU 要对此定时/计数器芯片进行写操作。

A_0 、 A_1 地址信号线。高位地址信号经译码产生 \overline{CS} 选片信号,决定了 8253 芯片所具有的地址范围。而 A_0 和 A_1 地址信号则经片内译码产生 4 个有效地址,分别对应于芯片内部 3 个独立的计数器(通道)和一个控制寄存器。具体规定如见表 8-2。

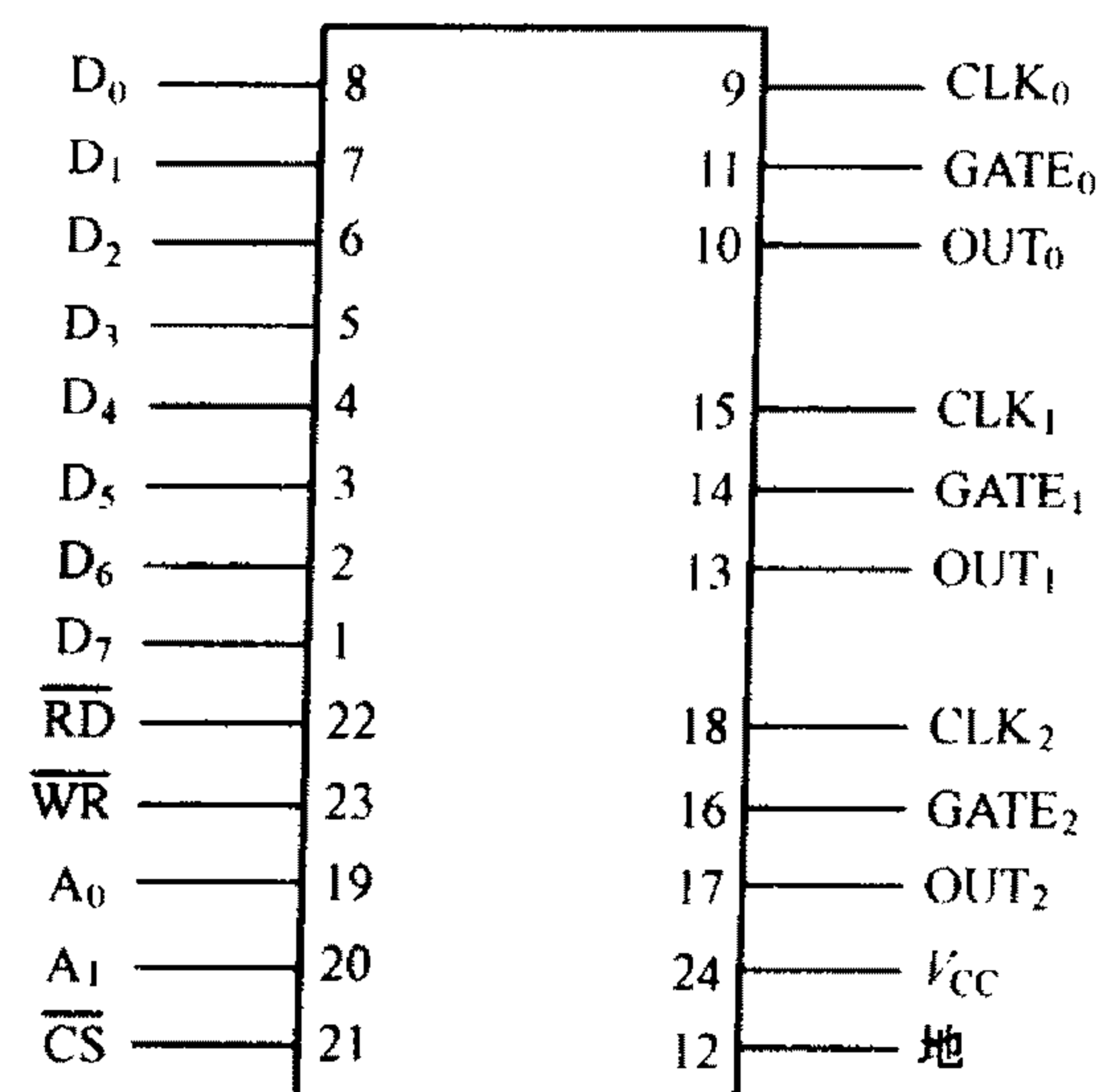


图 8.10 可编程定时/计数器 8253 外部引线排列图

表 8-2 计数器的选择

A_1	A_0	定义
0	0	选择计数器 0
0	1	选择计数器 1
1	0	选择计数器 2
1	1	选择控制寄存器

8253 的 I/O 引出线包括：

$CLK_0 \sim CLK_2$ 计数器的时钟信号输入端。计数器对此时钟信号进行计数,每个时钟脉冲将使计数器减 1。CLK 信号是计数器工作的计时基准,因此要求其频率要很精确。

$GATE_0 \sim GATE_2$ 门控信号,用于启动或禁止计数器的减 1 操作。多数情况下, $GATE = 1$ 时允许计数, $GATE = 0$ 时停止计数。但有时仅用 GATE 的上升沿启动计数,启动后则 GATE 的状态不再影响计数过程。

$OUT_0 \sim OUT_2$ 计数器输出信号。在不同的工作模式下,当计数器减为 0 时会在这些引出线上产生不同的输出波形。

2) 8253 的内部结构和工作原理

图 8.11 所示为 8253 的内部逻辑结构。它主要包括 3 个计数器通道、一个控制寄存器、

数据总线缓冲器及读/写逻辑电路。

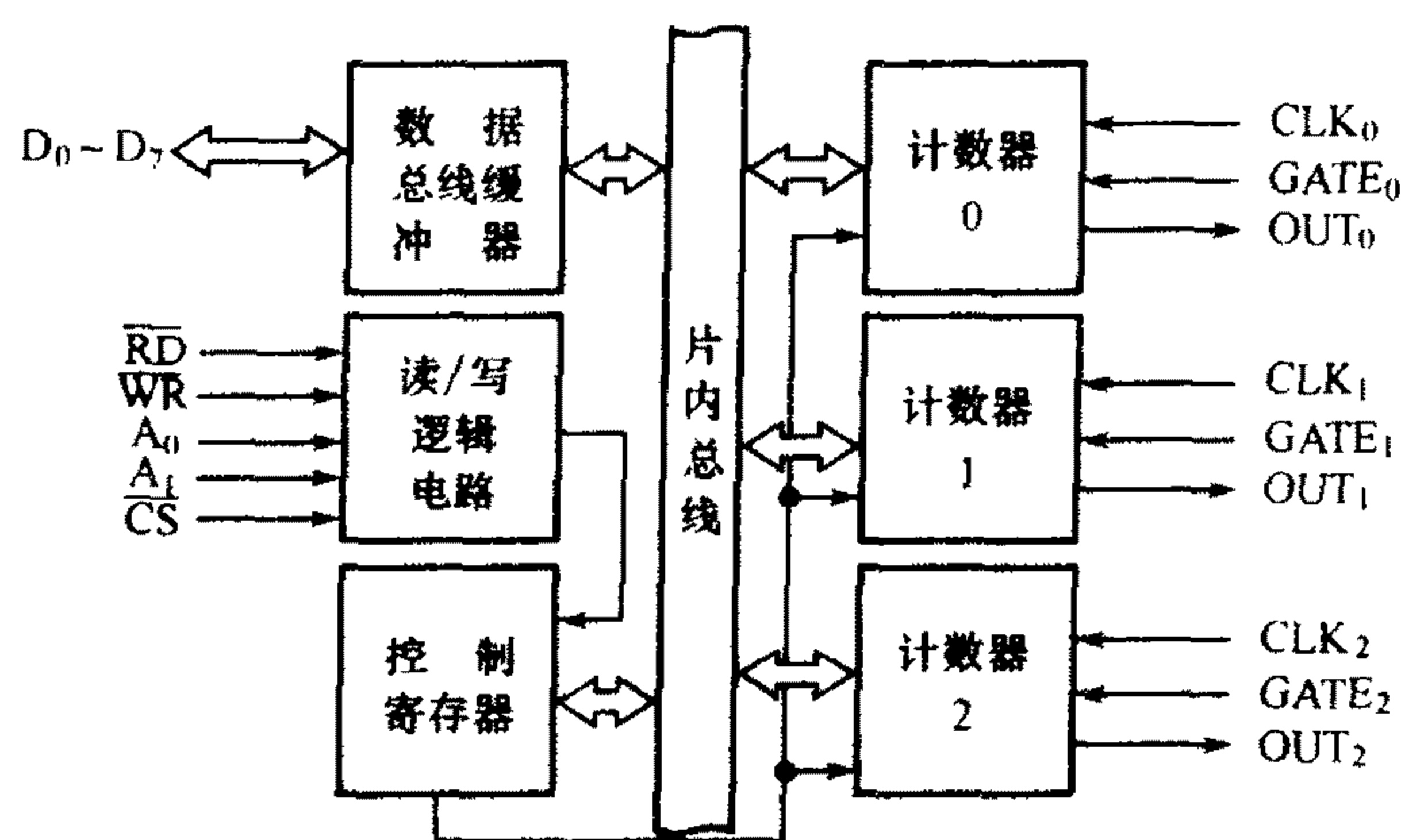


图 8.11 可编程定时/计数器 8253 的内部结构框图

(1) 计数器

计数器 0、计数器 1 和计数器 2 是 3 个相同的 16 位减 1 计数器，它们互相独立，可以分别按各自的方式进行工作。每个计数器都包括一个 16 位的初值寄存器、一个计数执行单元和一个输出锁存器。其工作过程如下：

当置入初值后，计数执行单元开始对输入脉冲 CLK 进行减 1 计数，在减到 0 时，从 OUT 端输出一个信号。整个过程可以重复进行。计数器既可按二进制计数，也可按十进制计数。另外，在计数过程中，计数器还受到门控信号 GATE 的控制。在不同的工作模式下，计数器的输入 CLK、输出 OUT 和门控信号 GATE 之间的关系将会不同。

(2) 控制寄存器

控制寄存器用来存放操作方式控制字。所谓可编程定时/计数器，就是允许向控制寄存器中写入所需的控制字，从而决定该计数器的工作方式。控制字是在 8253 初始化时用输出指令写入控制寄存器的。该寄存器只能写入，不能读出。

(3) 数据总线缓冲器

这是一个 8 位的双向三态缓冲器，用于 8253 和 CPU 数据总线之间连接的接口。CPU 通过该数据缓冲器对 8253 进行读/写。

(4) 读/写控制逻辑

在选片信号 \overline{CS} 有效的情况下，读/写控制逻辑从系统总线接收输入信号，经过逻辑组合，产生对各部分的控制信号。当选片信号 \overline{CS} 无效（高电平）时，数据总线缓冲器处于高阻态，8253 与总线断开，CPU 不能对其进行读/写操作。

3) 计数启动方法

8253 计数器的计数过程,可以由程序指令启动,称为软件启动;也可由外部电路信号启动,称为硬件启动。

(1) 软件启动

软件启动就是 CPU 用输出指令向计数器写入初值后就启动计数。但事实上,CPU 写入的计数初值只是写到了计数器内部的初值寄存器中,计数过程并未真正开始。写入初值后的第一个 CLK 信号只是将初值寄存器中内容送到了计数器中,而从第二个 CLK 脉冲的下降沿开始,计数器才真正进行减 1 计数。之后,每来一个 CLK 脉冲都会使计数器减 1,直到减到 0 时在 OUT 端输出一个信号。因此,从 CPU 执行输出指令写入计数初值到计数结束,实际的 CLK 脉冲个数比编程写入的计数初值 N 要多一个,即 $N+1$ 个。只要是用软件启动计数,这种误差便是不可避免的。

(2) 硬件启动

硬件启动是写入计数初值后并不启动计数,而是在门控信号 GATE 由低电平变高后,再经 CLK 信号的上升沿采样,之后在该 CLK 的下降沿才开始计数。由于 GATE 信号与 CLK 信号不一定同步,故在极端情况下,从 GATE 变高到 CLK 采样之间的延时可能会经历一个 CLK 脉冲宽度,因此在计数初值与实际的 CLK 脉冲个数之间也会有一个时钟脉冲的误差。

对于大多数的工作方式,计数器每启动一次只工作一个周期(即从初值减到 0),要想重复计数过程,则必须重新启动,这种方式称为不自动重复的计数方式。但有两种工作方式,一旦计数启动,只要门控信号 GATE 保持高电平,计数过程就会自动周而复始地重复下去,这时 OUT 端可以产生连续的波形输出,这种计数过程称为自动重复的计数方式。在自动重复计数方式下,达到稳定状态后,上面讲到的因启动造成的实际计数值和计数初值之间的误差就不再存在。

2. 8253 的工作方式

8253 共有六种不同的工作方式,在不同的工作方式下,计数过程的启动方式、OUT 端的输出波形都不一样,自动重复功能和 GATE 的控制作用以及写入新的计数初值对计数过程产生的影响也不相同。下面借助工作波形来分别说明这六种工作方式的计数过程。

1) 方式 0(计数结束中断方式)

方式 0 为软件启动、不自动重复计数的方式。

在这种方式下,在第一个写信号 \overline{WR} 有效时向计数器写入控制字 CW,之后其输出端 OUT 就变低电平。在第二个 \overline{WR} 有效时装入计数初值,然后经过一个 CLK 信号的上升沿和下降沿,初值进入计数器。当计数减到 0——计数结束后,OUT 输出变为高电平(波形如图 8.12 所示,图中的 CW 表示控制字, N 表示计数初值)。该输出信号可作为中断请求信号使用(“计数结束中断”方式的名称即来源于此)。

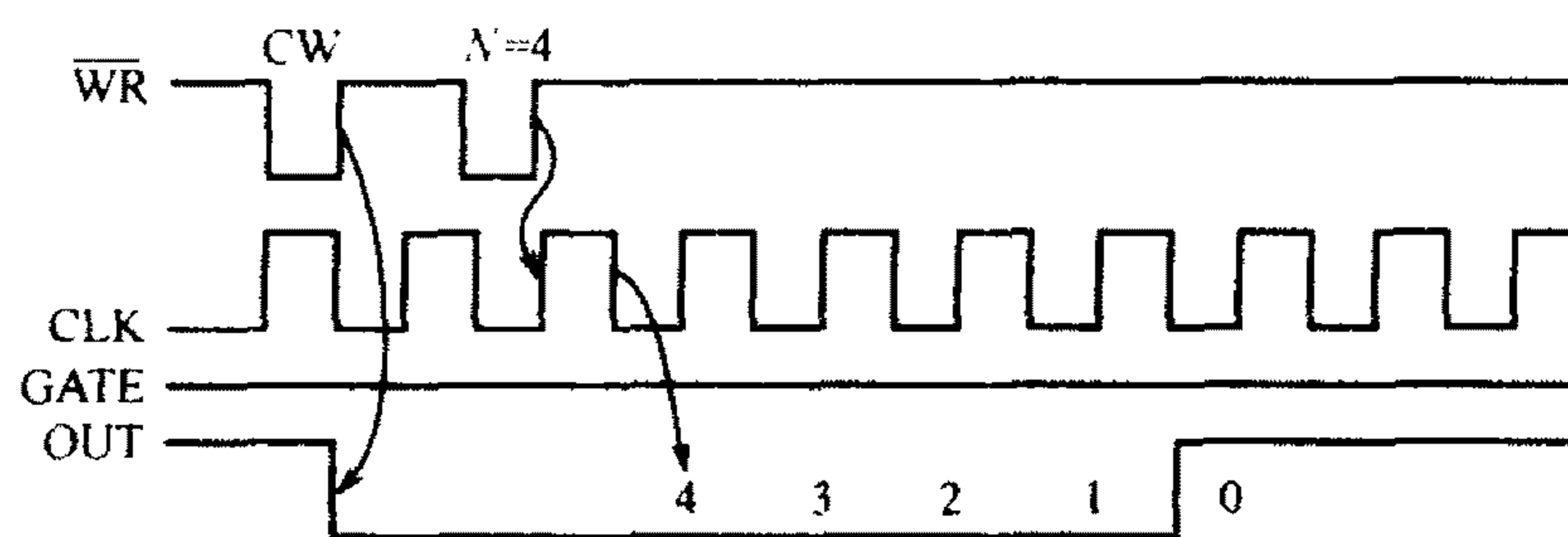


图 8.12 8253 方式 0 的工作波形

方式 0 的特点:

① 在整个计数过程中, GATE 端应始终保持为高电平。若 GATE 变低电平, 则暂停计数, 直到 GATE 变高电平后再接着计数。

② 在方式 0 下, 每写入一次计数初值只计数一个周期。计数结束后 OUT 端将保持高电平, 直到 CPU 再次写入新的计数初值。

③ 在计数过程中可随时修改计数初值, 即使原来的计数过程没有结束, 计数器也用新的计数初值重新计数。但如果新的计数初值是 16 位的, 则在写入第一个字节后停止原先的计数, 写入第二个字节后才开始以新的计数值重新计数。

2) 方式 1(可重复触发的单稳态触发器)

方式 1 是一种硬件启动、不自动重复的工作方式。当写入方式 1 的控制字后, OUT 端输出高电平。接着写入计数初值后, 计数器并不开始计数, 而是要等门控信号 GATE 出现由低到高电平的跳变(触发)后, 在下一个 CLK 脉冲的下降沿才开始计数, 此时 OUT 端立刻变为低电平。当计数结束后, OUT 端输出高电平。这样就可以从计数器的 OUT 端得到一个负脉冲, 负脉冲宽度为计数初值 N 乘以 CLK 的周期 T_{CLK} 。方式 1 的工作波形如图 8.13 所示。

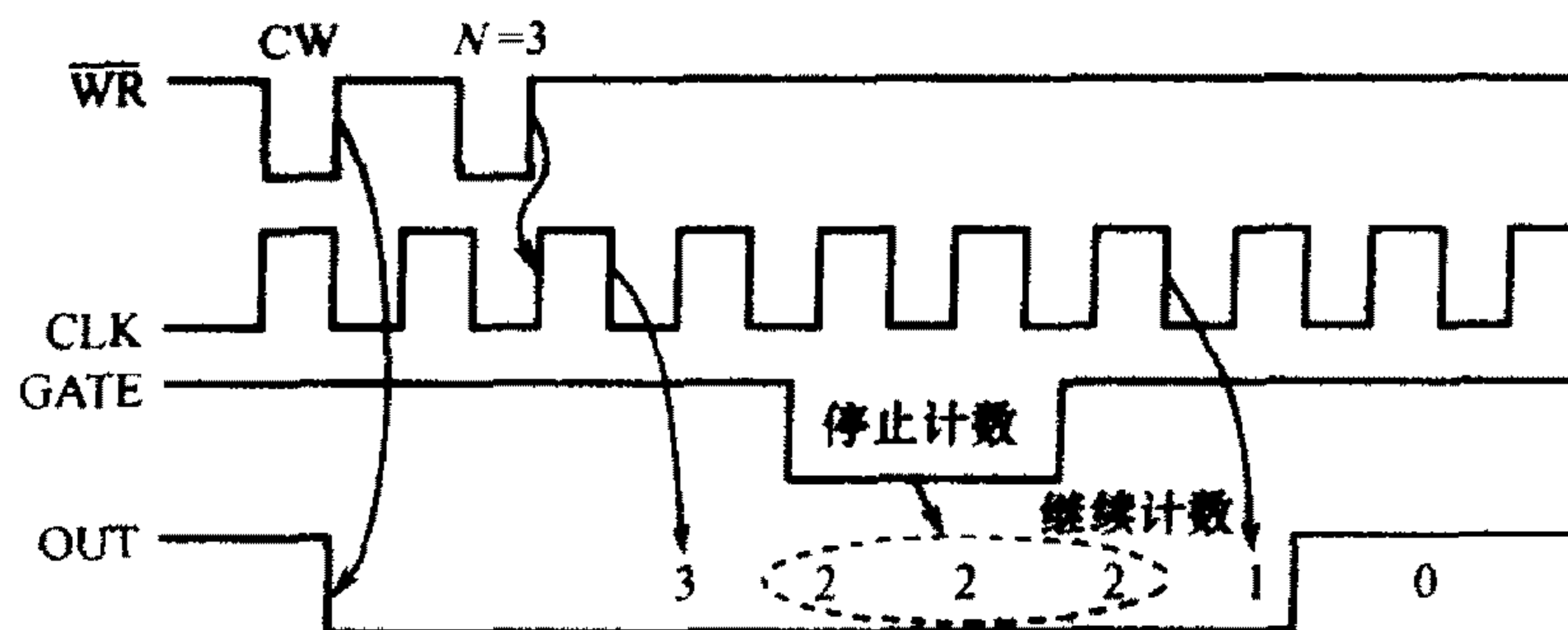


图 8.13 方式 1 时 GATE 信号的作用

方式 1 的特点:

① 计数过程一旦启动, GATE 端即使变低电平也不会影响计数。

② 可重复触发 当计数到 0 后,不用再次写入计数初值,只要用 GATE 的上升沿重新触发一次计数器,即可产生一个同样宽度的负脉冲。

③ 在计数过程中,若写入新的计数值,则本次计数过程的输出不受影响。本次计数结束后再次触发,计数器才开始按新的计数值进行计数,并按新值输出脉冲宽度。

④ 若在形成单个负脉冲的计数过程中,外部的 GATE 上升沿提前到来,则下一个 CLK 脉冲的上升沿使计数器重新装入计数初值,并紧接着在 CLK 的下降沿重新开始计数。这时的负脉冲宽度将会加宽,宽度为重新触发前的已有的宽度与新一轮计数过程的宽度之和。

3) 方式 2(频率发生器)

在这种方式下,计数器既可以用软件启动,也可以用硬件启动。若写入控制字和计数初值期间 GATE 一直为高电平,则在写入计数初值后的下一个 CLK 开始计数(即软件启动);若送计数初值时 GATE 为低电平,则要等到 GATE 信号由低电平变高电平时才启动(即硬件启动)。一旦计数启动,计数器可以自动重复工作。

在写入方式 2 控制字后,OUT 端变为高电平。假设此时 $GATE = 1$,则装入计数初值 N 后计数器从下一个 CLK 的下降沿开始计数,经过 $N - 1$ 个 CLK 周期后(此时计数值减为 1),OUT 端变为低电平,再经过一个 CLK 周期,计数值减到 0,OUT 又恢复为高电平。由于方式 2 下计数器可自动重复计数,因此在计数减到 0 后,计数器又自动装入计数初值,并开始新一轮的计数过程。这样,在 OUT 端就会连续输出宽度为 T_{CLK} 的负脉冲,其周期为 $N \times T_{CLK}$,即 OUT 端输出的脉冲频率为 CLK 的 $1/N$ 。所以方式 2 也称为分频器,分频系数就是计数初值 N 。可以利用不同的计数初值实现对 CLK 时钟脉冲进行 1 ~ 65 536 分频。方式 2 的工作波形如图 8.14 所示。

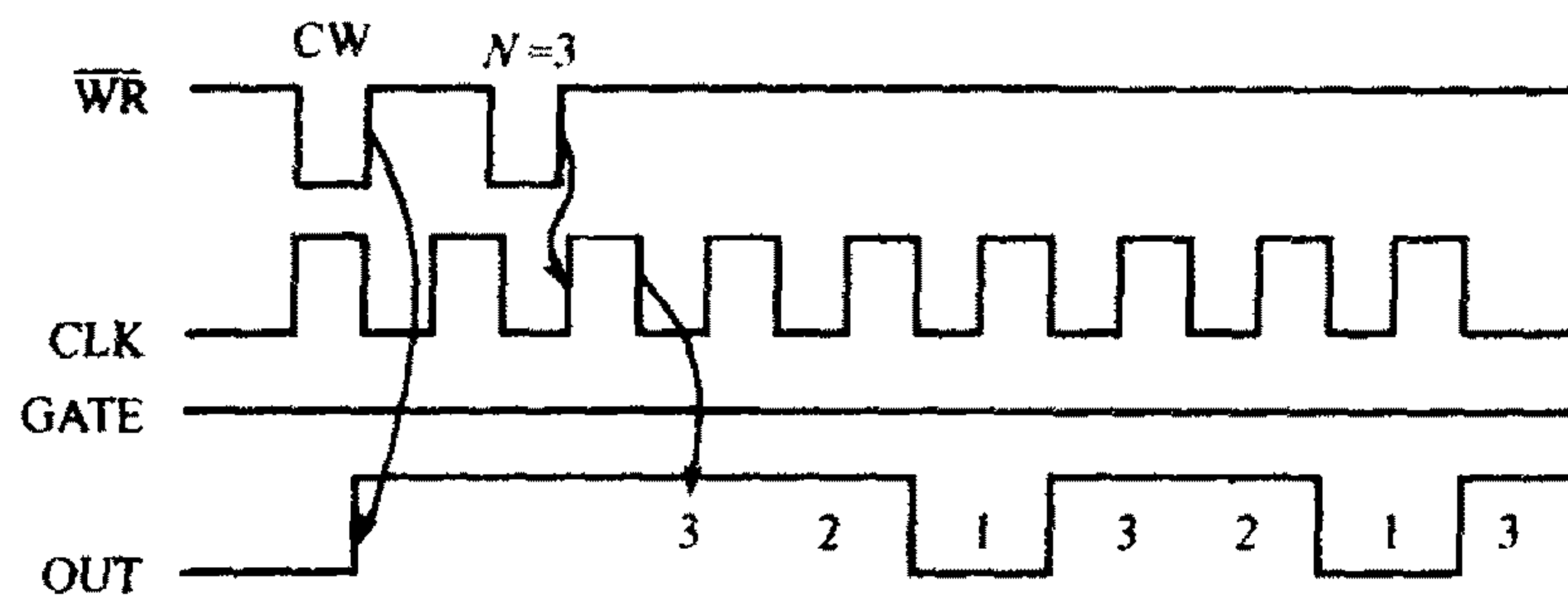


图 8.14 8253 方式 2 的工作波形

方式 2 的特点:

① 在方式 2 中,门控信号 GATE 可被用做控制信号:当 GATE 为低电平时,计数停止,强迫 OUT 输出高电平。当 GATE 变高电平后的下一个时钟下降沿,计数器又被置入初值从头开始重新计数,之后的过程就和软件启动相同。这个特点可用于实现计数器的硬件同步。

② 在计数过程中,若重新写入新的计数初值,则不影响当前的计数过程,而是在下一轮计数过程才按新的计数值进行计数。

方式 2 中,一个计数周期应包括 OUT 输出的负脉冲所占的那一个时钟周期。

4) 方式 3(方波发生器)

方式 3 和方式 2 类似,也有两种启动方式,也能够自动重复计数。只是计数到 $N/2$ 时,OUT 变为低电平,再接着计数到 0 时,OUT 又变为高电平,并开始新一轮计数。此时 OUT 端输出的波形不是负脉冲,而是方波。

当写入方式 3 的控制字 CW 后,OUT 端立刻变高电平。若此时 $GATE = 1$,则装入计数初值 N 后开始计数。如果装入的计数值 N 为偶数,则计数到 $N/2$ 时,OUT 变低,计完余下的 $N/2$ 后,OUT 又回到高电平。如此这般地自动重复下去,在 OUT 端就输出了周期为 $N \times T_{CLK}$ 的对称方波。这是理论上的说法,实际情况是:在方式 3 下,每来一个 CLK 脉冲,计数器减 2,减到 0 时 OUT 端变低电平(此时实际只计数 $N/2$ 个 CLK 脉冲);之后自动再将初值装入,仍是每个 CLK 脉冲计数器减 2,减到 0 时 OUT 端变高电平。工作波形如图 8.15 所示。

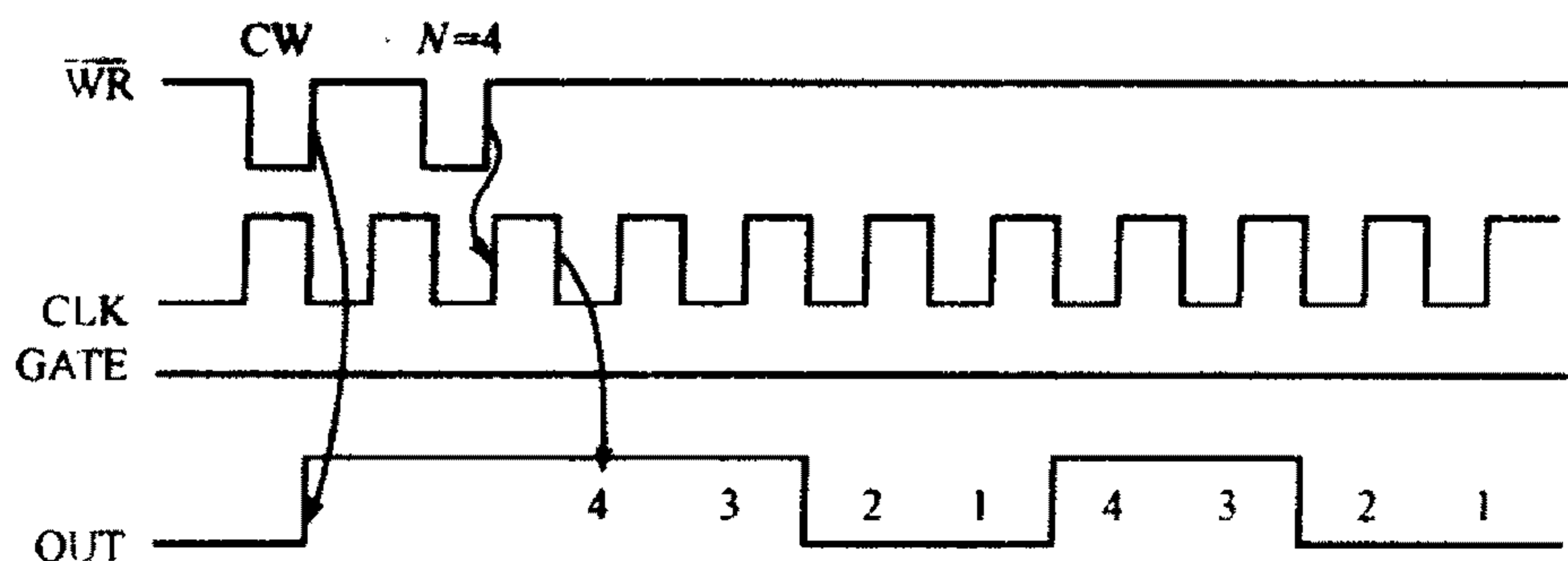


图 8.15 8253 方式 3 的工作波形

若 N 为奇数,则输出波形不对称,其中 $(N+1)/2$ 个时钟周期,OUT 为高电平,而另外 $(N-1)/2$ 个时钟周期,OUT 为低电平。

方式 3 的特点:

① 写入计数初值时,若 GATE 信号为低电平,则并不开始计数,OUT 端强迫输出高电平。直到 GATE 变为高电平后,才启动计数,输出对称方波。若计数过程中 GATE 变低电平,会立刻终止计数,且 OUT 端马上变高电平。当 GATE 恢复高电平后,计数器将重新装入计数初值,从头开始计数。

② 在计数过程中,若装入新的计数值,会在当前半周期结束时启用新的计数初值。当然,如果在改变计数初值后接着又发生硬件启动,则会立即以新计数值开始计数。

5) 方式 4(软件触发选通)

方式4为软件启动、不自动重复计数的方式。写入方式4控制字后,输出OUT立即变高电平。若 $GATE = 1$,则装入计数初值后计数立即开始。当计数结束时,由OUT输出一个宽度为 T_{CLK} 的负脉冲^①。方式4的工作波形如图8.16所示。

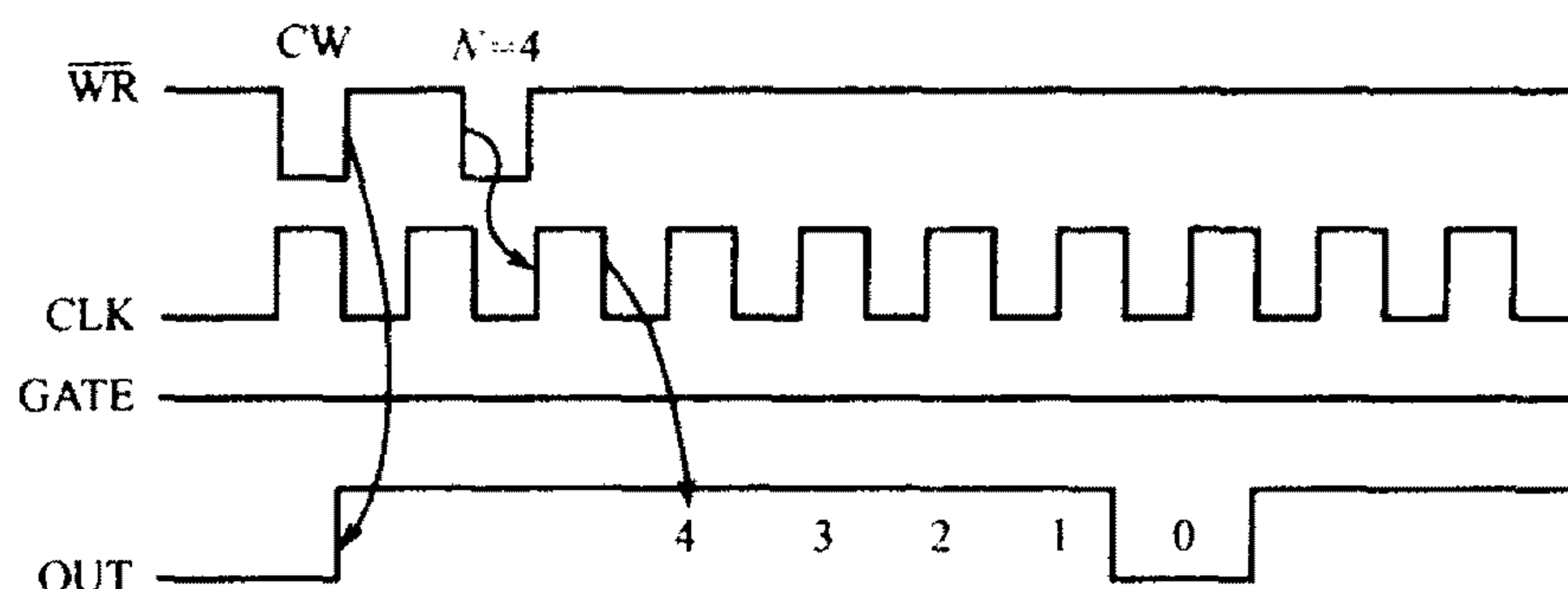


图 8.16 8253 方式4的波形

方式4的特点:

① 在这种方式下,每装入一次计数初值只进行一次计数,计数值减到0则停止,直到下一次装入初值又重新启动。此方式同样受GATE信号控制。只有当GATE为高电平时,计数才进行;当GATE为低电平时,则禁止计数。

② 如果在计数过程中装入新的计数值,则计数器从下一时钟周期开始按新的计数值重新开始计数。

请注意方式4与方式2下OUT端输出波形的不同。

6) 方式5(硬件触发选通)

方式5为硬件启动、不自动重复计数的计数方式。写入方式5控制字后,输出OUT变高电平。但此时即使GATE原来为高电平,也不能启动计数,而必须出现一个GATE上升沿跳变,计数才会开始。计数结束时OUT端送出一个宽度为 T_{CLK} 的负脉冲。之后,OUT又变高电平且一直保持到下一次计数结束。方式5的工作波形如图8.17所示。

方式5的特点:

① 由于方式5是硬件启动,同方式1一样,在启动后GATE变低电平,将不会影响计数过程的进行。但如果GATE又产生正跳变,则不论当前计数是否完成,又会给计数器重新装入初值,开始新一轮计数。

② 若在计数过程中改变计数初值,则新的计数值只写入到初值寄存器中,不影响当前计数,只在GATE发生正跳变后才以新的计数值计数。

^① 即计数到0后,OUT变低,再经过一个CLK周期,OUT又变为高。

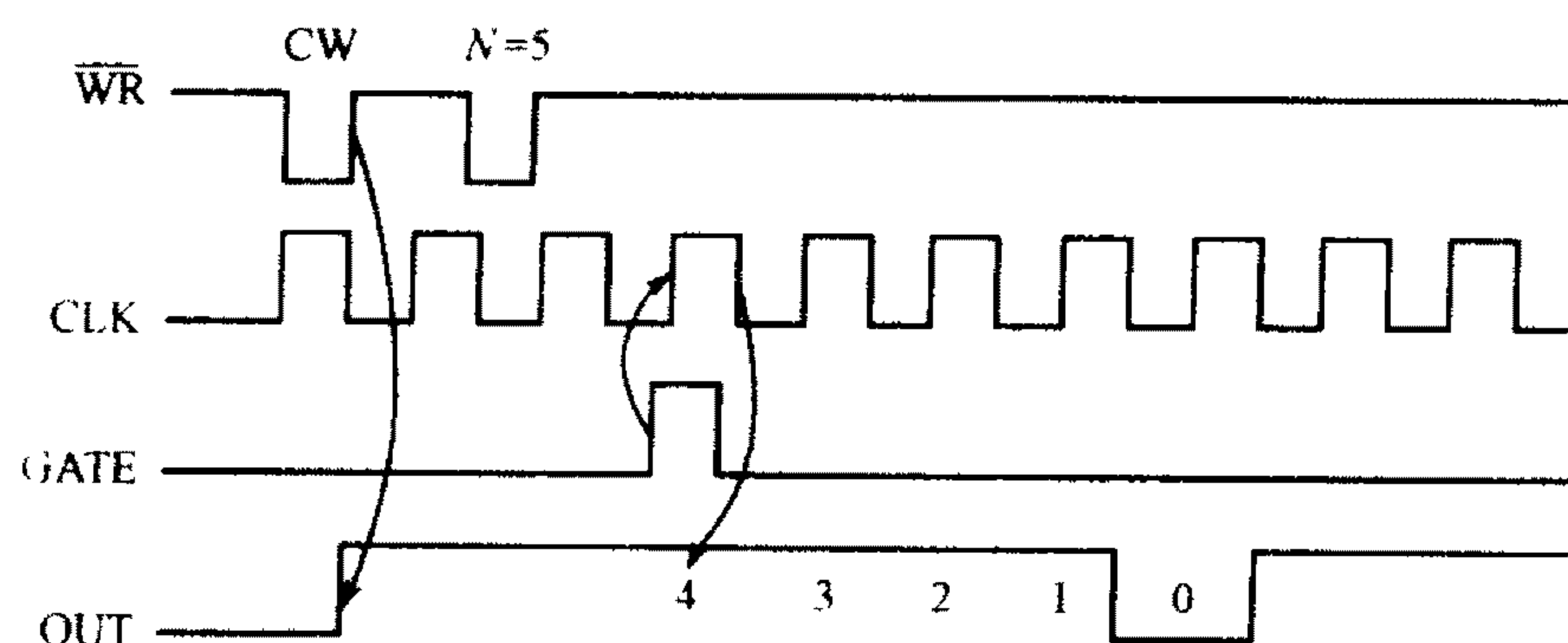


图 8.17 8253 方式 5 的工作波形

表 8-3 将 8253 计数器六种工作方式的特点综合在一起,以便于读者比较。

表 8-3 8253 计数器工作方式一览表

方式	启动计数	中止计数	自动重复	更新初值	输出波形
0	软件(写入初值)	$\text{GATE} = 0$	否	立即有效	延时时间可变的上升沿
1	硬件(GATE 正跳变)		否	下一轮有效	宽度为 $N \times T_{\text{CLK}}$ 的单一负脉冲
2	软件(写入初值) 硬件(GATE 正跳变)	$\text{GATE} = 0$	是	下一轮有效	周期为 $N \times T_{\text{CLK}}$ 、宽度为 T_{CLK} 的连续负脉冲
3	软件(写入初值) 硬件(GATE 正跳变)	$\text{GATE} = 0$	是	下半轮有效	周期为 $N \times T_{\text{CLK}}$ 的连续方波
4	软件(写入初值)	$\text{GATE} = 0$	否	下一轮有效	宽度为 T_{CLK} 的单一负脉冲
5	硬件(GATE 正跳变)		否	下一轮有效	宽度为 T_{CLK} 的单一负脉冲

3. 8253 的控制字

8253 必须先初始化才能正常工作,每个计数通道可分别初始化。CPU 通过指令将控制字写入可编程定时/计数器 8253 的控制寄存器,从而确定 3 个计数器分别工作于何种工作模式下。8253 的控制字具有固定的格式,如图 8.18 所示。

控制字的 D_0 位用来定义用户所使用的计数值是二进制数还是 BCD 数。因为每个计数器的字长都是 16 位,所以,如果采用二进制计数,则计数范围为 0000H ~ FFFFH;而如果用 BCD 计数,则计数范围为 0000 ~ 9999。由于计数器做减 1 操作,故当计数初值为 0000 时,对应的是最大计数值(二进制计数时为 65 536,十进制计数时为 10 000)。

在 8253 计数过程中,CPU 可随时读出其当前的计数值,而且不会影响计数器的工作。实现这种操作只需写入相应的控制字,此时控制字的 RL_1 、 RL_0 选择 00,即控制字格式为 SC_1

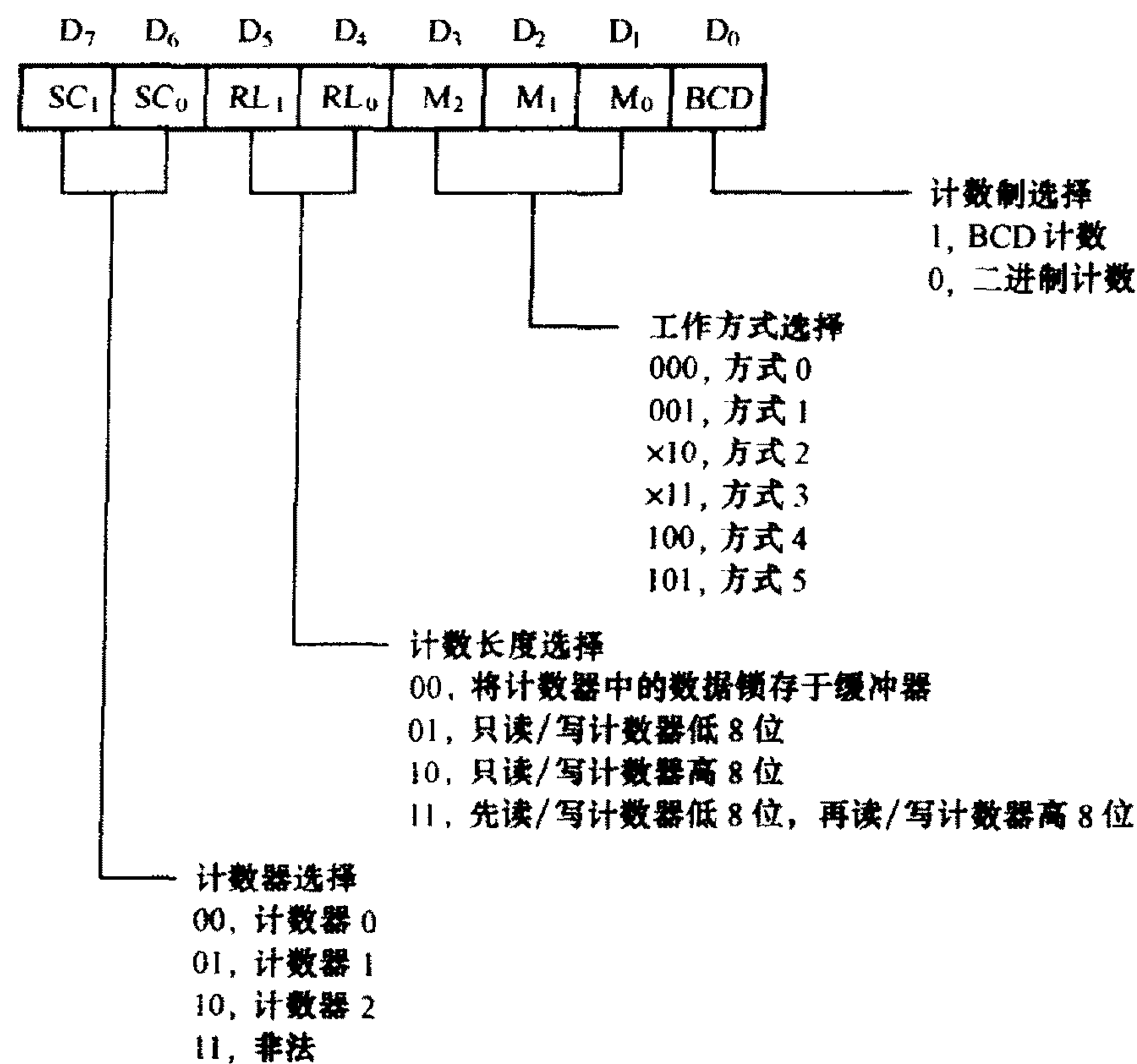


图 8.18 8253 方式控制字的格式

SC₀ 0 0 × × × ×。

4. 8253 的应用举例

1) 8253 与系统的连接

8253 共占用了 4 个接口地址, 地址范围由高位地址信号决定, 高位地址的译码输出接到片选端 \overline{CS} , A_0 和 A_1 分别接到系统总线的 A_0 、 A_1 地址信号线上, 用来寻址芯片内部的 3 个计数器及控制寄存器。信号 \overline{CS} 、 A_0 、 A_1 与读信号 \overline{RD} 、写信号 \overline{WR} 配合, 可以实现对 8253 的各种读/写操作。上述各信号的功能组合见表 8-4。

对 8253 的读/写操作要注意以下两点:

① 在向某一计数器写入计数初值时, 应与控制字中 RL_1 和 RL_0 的编码相对应。当编码为 01 或 10 时, 应只写入一个字节的计数值, 另一字节 8253 默认为 0; 当编码为 11 时, 则一定要装入两个字节的计数值, 且先写入低字节再写入高字节。若此时只写了一个字节就去写别的计数器的计数值, 则写入的字节将被解释为计数值的高 8 位, 从而产生错误。

② 8253 的计数器在计数过程中, 可读出其当前计数值。读出的方法有两种, 一是前面已讲到的在计数过程中读计数值的方法, 即写入 RL_1 和 RL_0 为 00 的控制字, 将选中的计数器的当前计数值锁存到相应缓冲器中, 而后, 利用读计数器操作——用两条输入指令即可把

16 位计数值读出;另一种方法是控制 GATE 门控信号使计数器停止计数,先写入控制字,规定好 RL_1 和 RL_0 的状态,也就是规定读一个字节还是读两个字节。若其编码为 11,则一定要读两次,先读出计数值低 8 位,再读出高 8 位。此时若只读一次,同样会出错。

表 8-4 各寻址信号组合功能

\overline{CS}	A_1	A_0	\overline{RD}	\overline{WR}	功能
0	0	0	1	0	写计数器 0
0	0	1	1	0	写计数器 1
0	1	0	1	0	写计数器 2
0	1	1	1	0	写控制寄存器
0	0	0	0	1	读计数器 0
0	0	1	0	1	读计数器 1
0	1	0	0	1	读计数器 2
0	1	1	0	1	无效

可编程定时/计数器 8253 可直接连接到系统总线上。图 8.19 所示就是 8253 与 80X86 系统总线连接的一个例子。图中地址总线信号 $A_{15} \sim A_2$ 经译码电路产生片选信号选中 8253。根据图中译码器的连接方式可知,该 8253 的接口地址范围为 FF04H ~ FF07H。FF04H

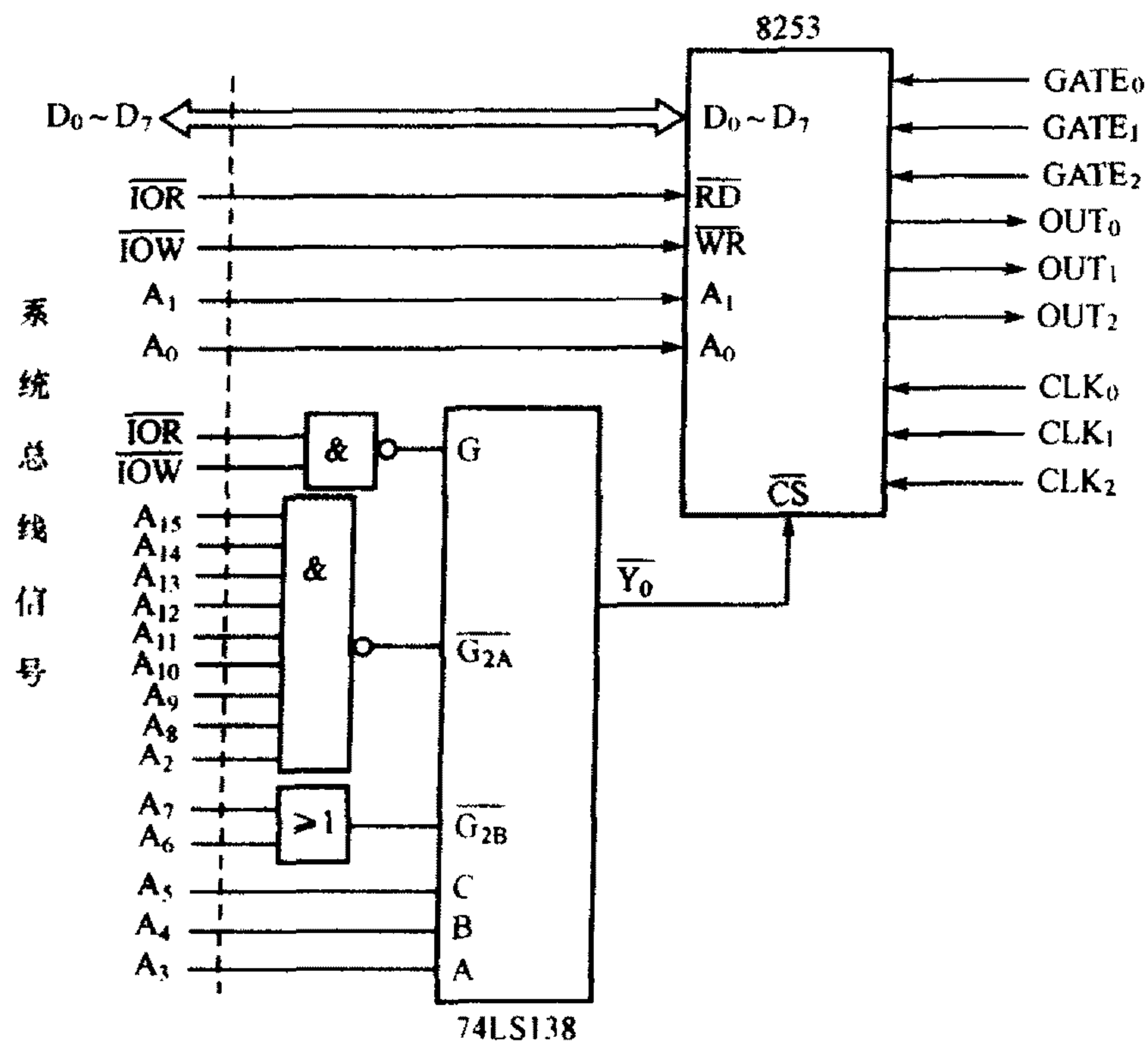


图 8.19 8253 与 80X86 系统总线的连接

对应于计数器 1, FF05H 对应于计数器 2, FF06H 对应于计数器 3, FF07H 对应于控制寄存器。

2) 8253 的初始化编程

8253 的初始化编程包括两部分,一是写各计数器的方式控制字,二是设置计数初值。由于 8253 每个计数器都有自己的地址,控制字中又有专门两位来指定计数器,这使得对计数器的初始化可按任何顺序进行。初始化的方法可以有两种:

① 以计数器为单位逐个进行初始化:即对某一个计数器,先写入方式控制字,接着写入计数初值(一个字节或两个字节)。先初始化哪一个计数器无关紧要,但对某一个计数器来说,则必须按照“方式控制字—计数值低字节—计数值高字节”的顺序进行初始化,如图 8.20 所示。

② 先分别写入 3 个计数器的方式控制字,再分别写入各计数器的计数初值。计数初值仍要按先低字节再高字节的顺序写入。

对以上两种初始化方法,读者可根据自己的习惯采用任意一种。

3) 8253 的应用举例

(1) 8253 在 IBM-PC 机中的应用

PC 机系统板上使用了一片 8253 定时/计数器。其计数器 1(CNT₁)用于 DRAM 的定时刷新,不能挪作它用。计数器 0(CNT₀)用来为系统的电子钟提供时间基准,它的输出端作为系统的时钟中断源,接到 8259 的 IR₀ 端,用以产生计时脉冲;计数器 2(CNT₂)主要用来作为机内扬声器的音频信号源,可输出不同频率的方波信号。图 8.21 是简化了的 IBM-PC 机的 8253 连接图,其接口地址采用了部分译码方式,占用外设地址 40H~5FH。在编程时,只使用这个地址范围中的 40H~43H。这 3 个计数器的输入时钟频率均为 1.19 MHz。

计数器 0 初始化为方式 3,产生周期的方波信号,其计数初值为 0000H,即最大计数值。根据方式 3 的工作原理可知,OUT₀ 输出方波信号的频率为 $1.19 \text{ MHz}/65536 = 18.2 \text{ Hz}$ 。由于 OUT₀ 在系统主板上与 8259 的中断请求输入线 IR₀ 相连接,所以每秒将会产生 18.2 次中断请求,该中断请求用于维护系统的日历钟。

计数器 1 初始化为方式 2,计数初值取 18, $18/1.19 \text{ MHz} \approx 15 \mu\text{s}$,即每 15 μs 对动态存储器刷新一次。

计数器 2 初始化为方式 3,控制扬声器发出频率为 1 kHz 的声音,故取计数初值为 1190。在 IBM-PC 中,要使扬声器发声,还必须使 8255 的 PB₁ 和 PB₀ 输出高电平。8255 的 B 口地址为 61H。

IBM-PC 中 8253 的初始化程序如下:

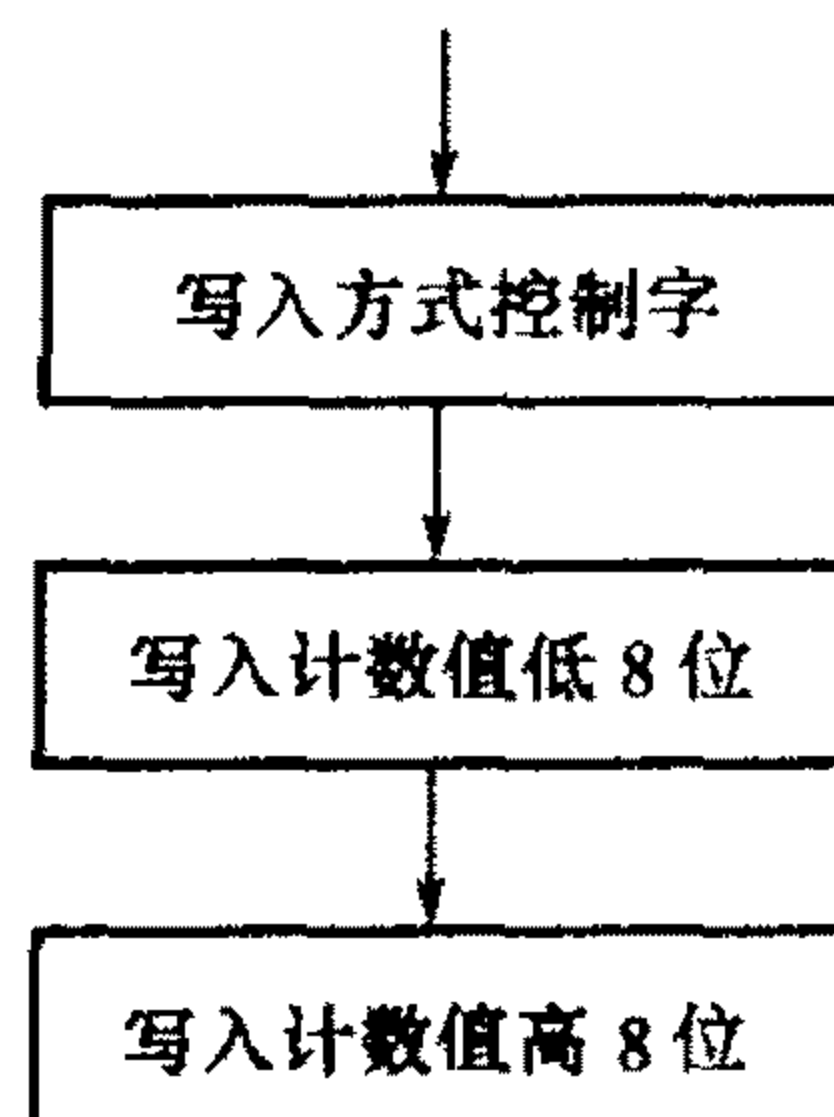


图 8.20 8253 方式控制字和计数初值的写入顺序

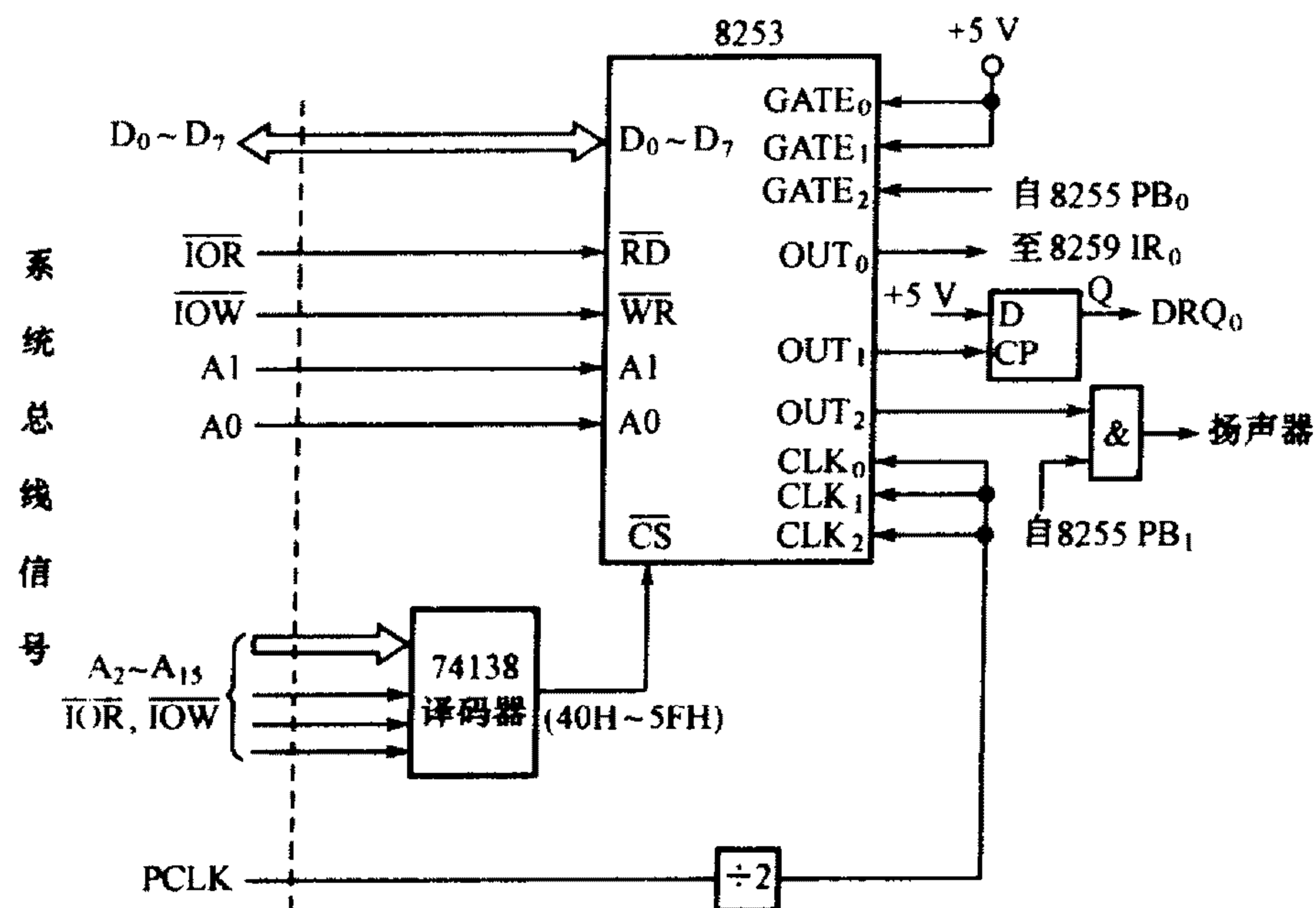


图 8.21 IBM-PC 机中的 8253 的连接简图

;CNT₀ 初始化

```

MOV AL, 36H      ;选择计数器 0, 写双字节计数值, 方式 3, 二进制计数
OUT 43H, AL      ;控制字写入控制寄存器
MOV AL, 0        ;选最大计数值(65 536)
OUT 40H, AL      ;写低 8 位计数值
OUT 40H, AL      ;写高 8 位计数值

```

;CNT₁ 初始化

```

MOV AL, 54H      ;选择计数器 1, 低 8 位单字节计数值, 方式 2, 二进制计数
OUT 43H, AL
MOV AL, 18
OUT 41H, AL      ;计数值写入计数器 1

```

;CNT₂ 初始化

```

MOV AL, 0B6H     ;选择计数器 2, 双字节计数值, 方式 3, 二进制计数
OUT 43H, AL
MOV AX, 1190
OUT 42H, AL      ;送低字节到计数器 2
MOV AL, AH       ;(AL) ← 高字节计数值
OUT 42H, AL      ;高 8 位计数值写入计数器 2
IN AL, 61H       ;读 8255 的 B 口
MOV AH, AL       ;将 B 口内容保存

```

```

OR AL, 03          ;使 PBO = PB1 = 1
OUT 61H, AL        ;使扬声器发声
:
MOV AL, AH         ;恢复 8255B 口状态
OUT 61H, AL

```

(2) 脉冲发生器

电路如图 8.19 所示。3 个计数器 CLK 频率均为 2 MHz。要求计数器 0 在定时 100 μs 后产生中断请求;计数器 1 用于产生周期为 10 μs 的对称方波;计数器 2 每 1 ms 产生一个负脉冲。编写 8253 的初始化程序。

根据要求可知,计数器 0 应工作于方式 0,计数初值 = $100 \mu\text{s} / 0.5 \mu\text{s} = 200$ (CLK 的周期 = 0.5 μs);计数器 1 应工作于方式 3,计数初值 = $10 \mu\text{s} / 0.5 \mu\text{s} = 20$;计数器 2 应工作于方式 2,计数初值 = $1 \text{ ms} / 0.5 \mu\text{s} = 2000$ 。以下是 8253 的初始化程序。

```

INIT8253: MOV DX, 0FF07H
          MOV AL, 10H          ;计数器 0,只写计数值低 8 位,方式 0,二进制计数
          OUT DX, AL
          MOV AL, 56H          ;计数器 1,只写计数值低 8 位,方式 3,二进制计数
          OUT DX, AL
          MOV AL, 0B4H         ;计数器 2,先写高 8 位,再写低 8 位,方式 2,二进制计数
          OUT DX, AL
          MOV DX, 0FF04H
          MOV AL, 200          ;计数器 0 的计数初值
          OUT DX, AL
          MOV DX, 0FF05H
          MOV AL, 20           ;计数器 1 的计数初值
          OUT DX, AL
          MOV DX, 0FF06H
          MOV AX, 2000         ;计数器 2 的计数初值
          OUT DX, AL
          MOV AL, AH
          OUT DX, AL
          ...

```

本例中,8253 通过对外部输入时钟信号的计数,可以达到计数和定时两种应用目的。门控信号 GATE 提供了从外部控制计数器的能力。当一个计数器计数或定时长度不够时,还可以把两个、三个计数器串联起来使用,即一个计数器的输出 OUT 作为下一个计数器的外部时钟 CLK 输入,甚至可将两个 8253 串起来使用。

8.2.2 可编程并行输入/输出接口 8255

并行接口一次可以同时传送一个数据的所有位。对一个具体的并行接口来说,其数据传送方向有两种,一是单向传送(只作为输入口或输出口),另一种是双向传送(既可作为输入口,也可作为输出口)。并行接口可以很简单(如锁存器或三态门),也可以很复杂(如可编程并行接口芯片),功能完善的并行接口中一般都包括输入/输出数据寄存器、控制寄存器(存放控制命令)、状态寄存器(保存当前工作状态)和总线缓冲器等部件。

8255 是 Intel 公司为 80X86 系列 CPU 配套的可编程并行接口芯片。8255 的通用性较强,使用灵活,是一种典型的可编程并行接口。

1. 8255 的引脚及结构

1) 外部引脚及功能

8255 的外部引脚如图 8.22 所示。其中与系统总线连接的引出线包括:

$D_0 \sim D_7$ 双向数据信号线,用来传送数据和控制字。

\overline{RD} 读信号线,低电平有效。与其他信号线一起实现对 8255 接口的读操作。通常接系统总线的 \overline{IOR} 信号。

\overline{WR} 写信号线,低电平有效。与其他信号一起实现对 8255 的写操作。通常接系统总线的 \overline{IOW} 。

\overline{CS} 片选信号线,低电平有效。当系统地址信号经译码产生低电平时选中 8255 芯片,即可对 8255 进行操作。

A_0, A_1 端口地址选择信号线。

8255 的内部包括 3 个独立的输入/输出端口(A 口、B 口和 C 口)以及一个控制寄存器。 A_0, A_1 地址信号经片内译码可产生 4 个有效地址,分别对应 A 口、B 口、C 口和内部控制寄存器。具体规定表 8-5 所示。

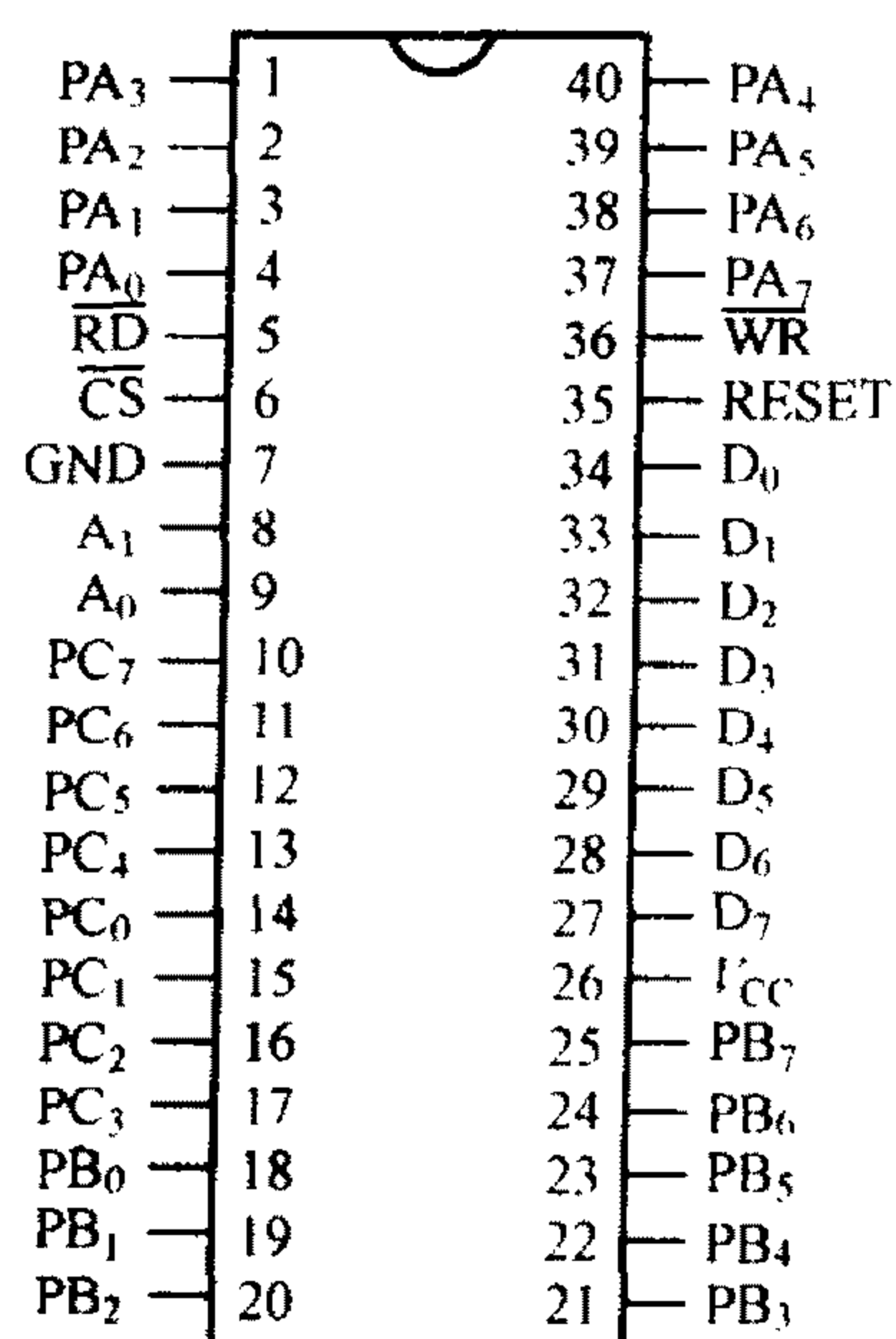


图 8.22 8255 的外部引脚图

表 8-5 A、B、C 口的选择

A_1	A_0	定义
0	0	选择 A 口
0	1	选择 B 口
1	0	选择 C 口
1	1	选择控制寄存器

在实际使用中, A_0 、 A_1 通常接系统总线的 A_0 和 A_1 , 它们与 \overline{CS} 一同决定 8255 的接口地址。

RESET 复位输入信号。通常接系统的复位 RESET 端。当它为高电平时使 8255 复位, 复位后, 8255 的 A 口、B 口和 C 口均被初始化为输入状态。

8255 与外设连接的 I/O 引出线包括:

PA0 ~ PA7 A 口的 8 根输入/输出信号线。这 8 根线可被设置为仅输入、仅输出或双向(同时为输入或输出)方式, 由软件编程来决定。

PB₀ ~ PB₇ B 口的 8 根输入/输出信号线。可用软件编程设置这 8 根线是输入还是输出。

PC₀ ~ PC₇ C 口的 8 根线, 根据不同的工作方式可作为数据的输入或输出, 也可以用做控制信号的输出或状态信号的输入, 具体使用方法将在本节稍后介绍。

2) 8255 的内部结构

8255 的内部结构如图 8.23 所示。它由以下几个部分构成:

(1) 数据端口

8255 有 A 口、B 口、C 口 3 个 8 位数据端口, 可以通过编程把它们分别指定为输入口或输出口。A 口和 B 口的输入/输出都具有数据锁存能力。C 口仅输出有锁存能力, 而输入没有锁存能力。A、B、C 3 个口作输出时, 其输出锁存器的内容可以由 CPU 用输入指令读回。在使用中, A、B、C 3 个口可作为 3 个独立的 8 位数据输入/输出口; 也可只将 A、B 口作为数据输入/输出口, 而将 C 口的各位作为 A 口和 B 口与外设联络用的状态输入或选通控制输出。C 口的主要特点是可以对其按位进行操作。

(2) A 组和 B 组控制电路

8255 把 A、B、C 3 个口分为 A 组和 B 组, 并分别由两组控制电路进行控制。从图 8.23 中可以看到, 这两组控制电路一方面接收读/写控制逻辑的读/写命令, 另一方面接收由数据总线输入的控制字, 分别控制 A 组和 B 组的读/写操作和工作方式。A 组包括 A 口的 8 位和 C 口的高 4 位($PC_4 \sim PC_7$), B 组包括 B 口的 8 位和 C 口的低 4 位($PC_0 \sim PC_3$)。编程写入的控制字输入到内部控制寄存器, 控制 A 组和 B 组的工作方式。

(3) 读/写控制逻辑

读/写控制逻辑负责管理 8255 的数据传送。它把 A_0 、 A_1 、 \overline{CS} 、 \overline{RD} 、 \overline{WR} 和 RESET 信号进行逻辑组合, 形成相应的控制命令, 发送到 A 组和 B 组控制电路, 以控制信息的传送。

(4) 数据总线缓冲器

这是一个三态双向 8 位数据缓冲器, 8255 通过它和系统的数据总线相连, 传递控制字、数据和状态信息。

图 8.24 给出了 8255 各引出线与系统总线的连接示意图。

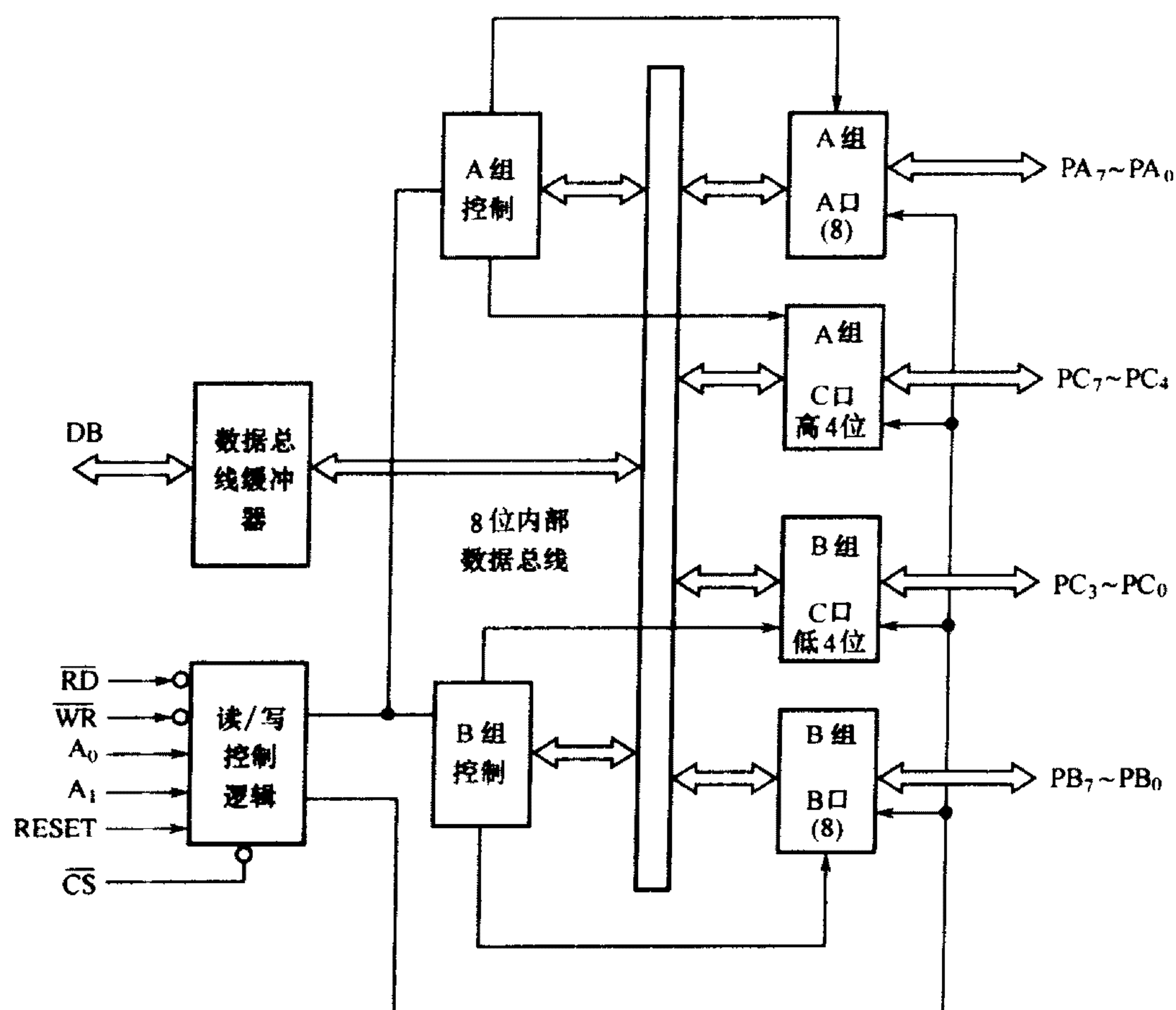


图 8.23 8255 内部结构框图

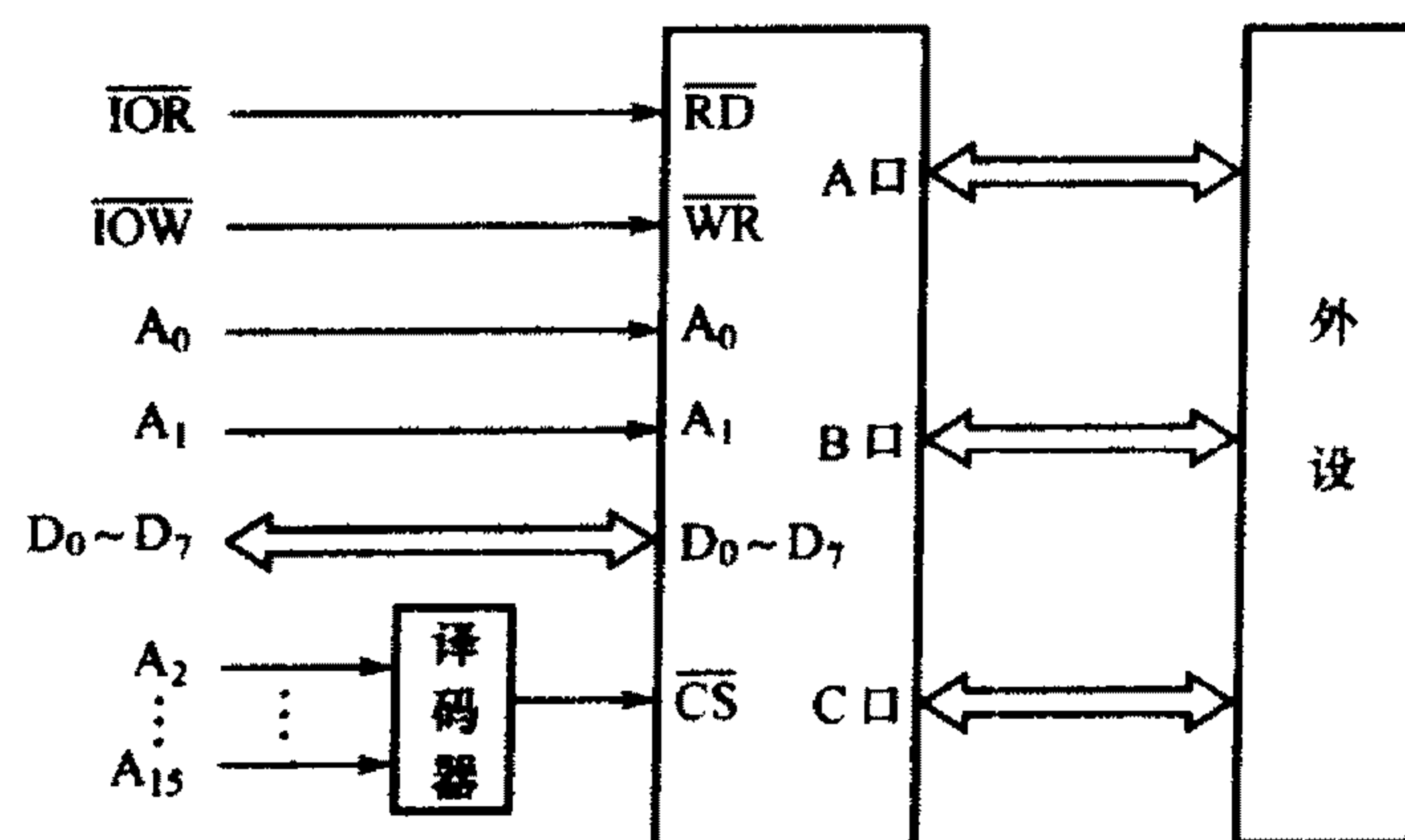


图 8.24 8255 与系统总线的连接示意图

2. 8255 的工作方式

8255 有三种基本的工作方式：方式 0、方式 1 和方式 2。其中 A 口可以工作在方式 0、方

式 1 或方式 2, B 口和 C 口只能工作于方式 0 或方式 1。3 个端口的工作方式可通过软件编程来设定。

1) 方式 0

方式 0 又称为基本输入/输出方式。方式 0 下各端口的形态如图 8.25 所示。

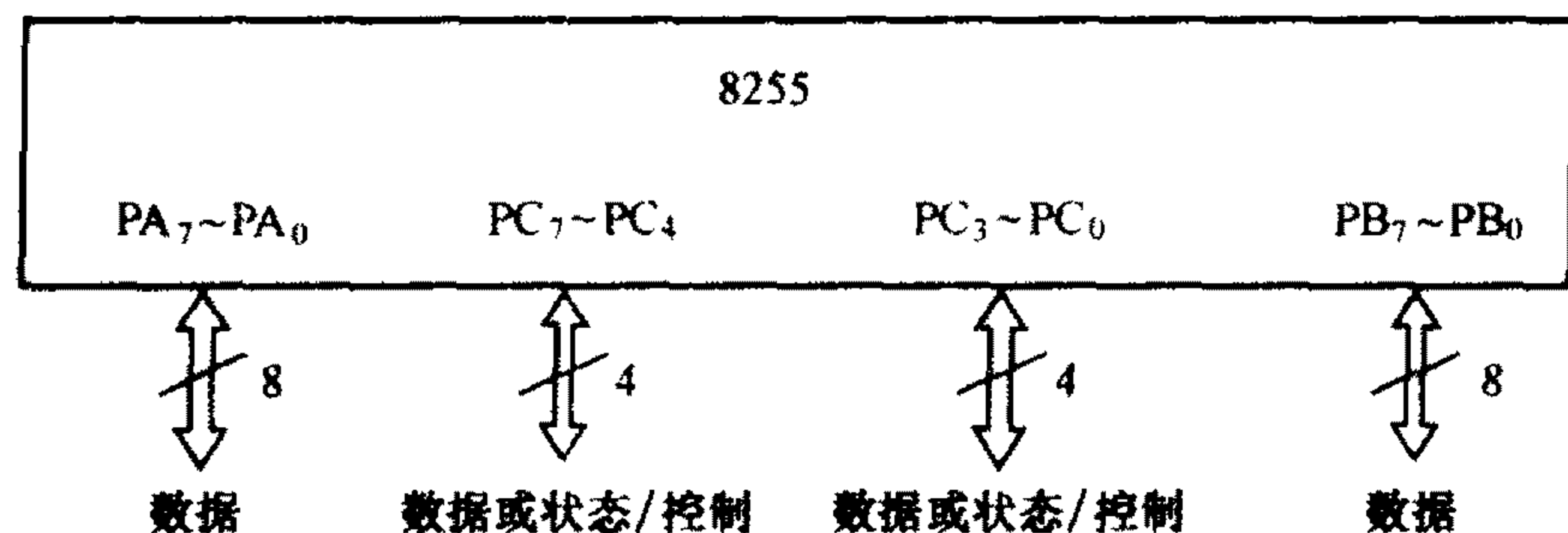


图 8.25 方式 0 下的端口形态

在这种方式下:

① A 口、C 口的高 4 位, B 口、C 口的低 4 位可分别定义为输入或输出, 它们互相独立, 故共有 16 种不同的组合。例如, 可定义 A 口和 C 口高 4 位为输入口, B 口和 C 口低 4 位为输出口, 或 A 口为输入, B 口、C 口高 4 位、C 口低 4 位为输出, 等等。

② 定义为输出的口均有锁存数据的能力, 而定义为输入的口则无锁存能力。

③ 在方式 0 下, C 口有按位进行置位和复位的能力。有关 C 口的按位操作见后面的叙述。

方式 0 最适合用于无条件传送方式, 由于传送数据的双方互相了解, 所以既不需要发控制信号给对方, 也不需要查询对方状态, 故 CPU 只需直接执行输入/输出指令便可将数据读入或写出。在无条件传送方式下, A、B、C 3 个口的全部 24 位都可以用做数据线。

方式 0 也能用于查询工作方式, 查询工作方式需要通信双方互相了解对方的状态, 但方式 0 由于没有规定固定的应答信号, 这时常将 C 口的高 4 位(或低 4 位)定义为输入, 用来接收外设的状态信号; 而将 C 口的另外 4 位定义为输出, 用来产生控制信号。此时的 A、B 口可用来传送数据。

2) 方式 1

这种方式也称为选通的输入/输出方式。在这种工作方式下, A、B、C 3 个口被分为两组。A 组包括 A 口和 C 口的高 4 位, A 口可由编程任意设定为输入口或输出口, C 口的高 4 位则用来作为 A 口输入/输出操作的控制和同步信号; B 组包括 B 口和 C 口的低 4 位, B 口可由编程任意设定为输入口或输出口, C 口的低 4 位则用来作为 B 口输入/输出操作的控制和同步信号。A 口和 B 口的输入数据和输出数据都被锁存。

为便于说明如何利用 C 口提供的控制和同步信号来进行输入/输出,下面我们分别以 A 口、B 口都设定为输入口(或都设定为输出口)来加以解释。

(1) A 口、B 口都设定为输出口

此时要利用 C 口的 6 根线作为 A 口和 B 口的控制信号。图 8.26 给出了控制信号线的安排。注意,这种安排是固定的,不允许改变: A 口使用 PC_3 、 PC_6 和 PC_7 ,而 B 口用 PC_0 、 PC_1 和 PC_2 。由 C 口提供的控制信号功能如下:

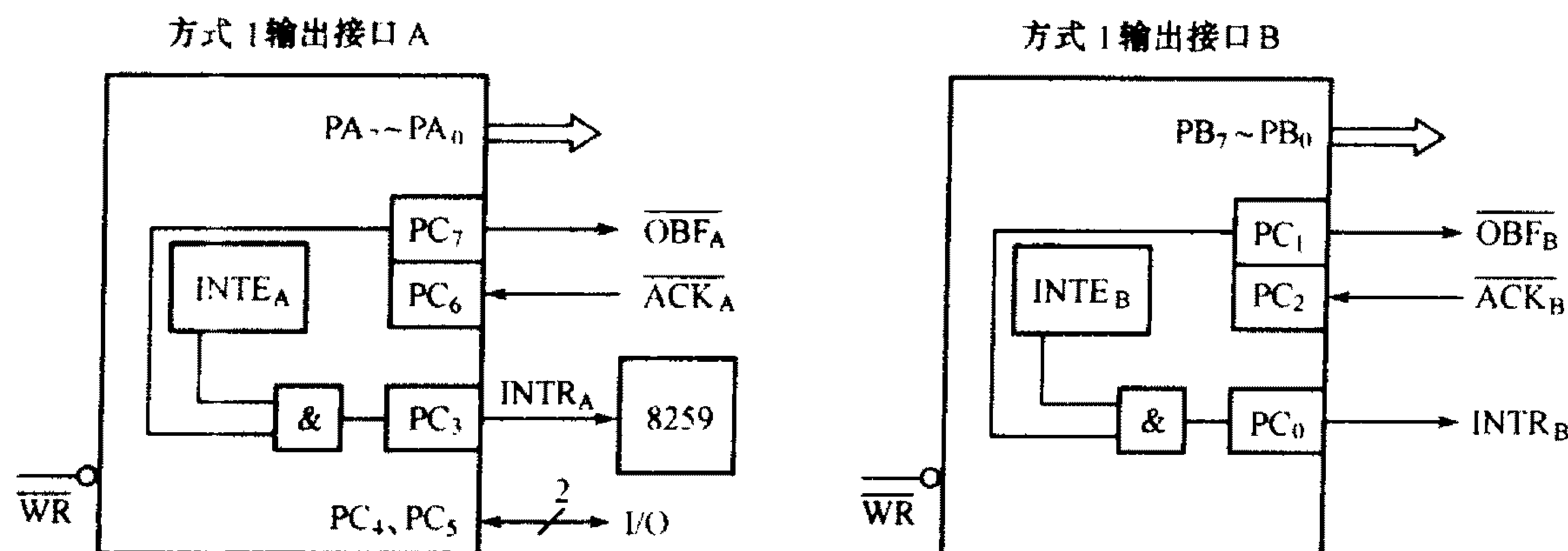


图 8.26 方式 1 下 A、B 口设定为输出时的控制信号定义

① 输出缓冲器满信号 \overline{OBF} , 输出, 低电平有效。该信号通知外设, 在规定的端口上已有一个有效数据, 外设可以从该端口读走数据。

② 外设响应信号 \overline{ACK} , 输入, 低电平有效。有效时, 表示外设已从该端口取走数据 \overline{ACK} , 信号有效时还使 $\overline{OBF} = 1$ 。

③ 中断请求信号 $INTR$, 输出, 高电平有效。当外设取走一个数据后, 其 \overline{ACK} 信号的上升沿会引发一个有效的 $INTR$ 信号, 该信号用于通知 CPU 可以再输出下一个数据。 $INTR$ 的有效条件为 $\overline{OBF} = 1$, $\overline{ACK} = 1$, $INTE = 1$ 。在采用中断输入/输出方式时, $INTR$ 可以作为中断请求信号向 CPU 申请中断。

④ 中断允许状态 $INTE$ 。8255 内部为每一组都设置了一个中断允许位 $INTE$ (如图 8.26 所示), 当 $INTE$ 为高电平, 且 \overline{OBF} 也变高电平时, 将产生有效的 $INTR$ 信号。 $INTE$ 由 PC_6 (A 口) 或 PC_2 (B 口) 的置位/复位来控制。

$INTR$ 是否输出高电平是由 $INTE$ 和 \overline{ACK} 信号共同决定的。以 A 口为例, 当 CPU 向接口写数据时 (执行一条 OUT 指令), 在 IOW 有效期间将数据锁存于芯片的数据缓冲器中, 之后在 IOW 的上升沿使 $\overline{OBF} = 0$ (PC_7 端输出负脉冲), 通知外部设备, A 口已有数据准备好。一旦外设将数据取走, 它就送出一个有效的 \overline{ACK} 脉冲, 该脉冲使 $\overline{OBF} = 1$, 若 $INTE$ 也为高电平, 就会在 PC_3 端产生一个有效的 $INTR$ 信号。该信号可接到中断控制器 8259 的中断请求线 IR 端,

进而向 CPU 提出中断请求。CPU 响应中断后,向接口写入下一个数据,同样由 $\overline{\text{IOW}}$ 将数据锁存,当数据锁存并由信号线输出,8255 就去掉 INTR 信号并使 $\overline{\text{OBF}}$ 有效,重复上述过程。方式 1 下的整个输出过程可参考图 8.27 所示的工作时序图。

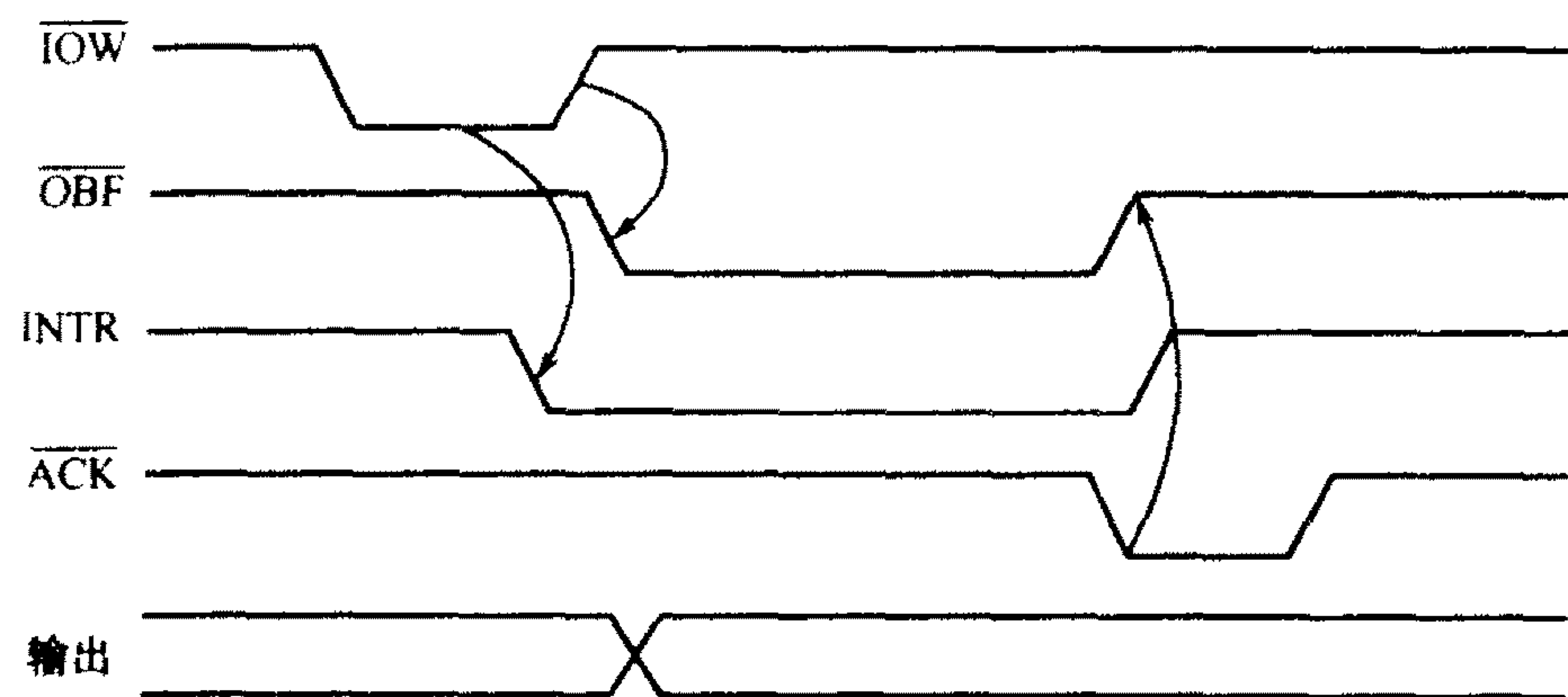


图 8.27 工作方式 1 下的数据输出时序

在这里提醒读者注意,当 A、B 两个口都设置为方式 1 输出时,仅使用了 C 口的 6 根线作为控制信号线,余下的 PC_4 、 PC_5 这时可作为 2 个 1 位的输入/输出线使用,可以用程序指定它们的传送方向是输入还是输出,而且也可通过位控操作对它们进行置位或复位。若 A、B 两个口中仅有一个口工作在方式 1,则只需使用 C 口的 3 根线作为控制信号,这时余下的 5 根线也可按照上面所说的方式工作。

(2) A 口、B 口都设定为输入口

与方式 1 下 A、B 口都设置为输出类似,A、B 口要实现选通输入,同样要利用 C 口提供的控制信号。在 A、B 都设置为输入口时,C 口的安排如图 8.28 所示。C 口的 PC_3 、 PC_4 和 PC_5 用于控制 A 口的数据输入, PC_0 、 PC_1 和 PC_2 用于控制 B 口的数据输入。C 口各控制信号功能如下:

① $\overline{\text{STB}}$ 输入选通信号,输入,低电平有效。它由外部设备提供,外设用 $\overline{\text{STB}}$ 信号将数据锁存于 8255 的输入数据缓冲器中。

② IBF 输入缓冲器满信号,输出,高电平有效。当 IBF 有效时,表示 8255 的输入数据缓冲器中有一个数据尚未被 CPU 读走。外设可根据此信号来决定是否能送下一个数据。可以将它看成是 $\overline{\text{STB}}$ 的应答信号。

③ INTR 中断请求信号,输出,高电平有效。当 $\overline{\text{STB}} = 1$ 时会使 IBF 变为高电平,从而产生有效的 INTR 信号,向 CPU 提出中断请求。也就是说,当外设将数据锁存于输入数据缓冲器中,且当前又允许中断请求发生时,就会在 INTR 线上产生一个中断请求。

④ 中断允许状态 INTE 在方式 1 下输入数据时,INTR 同样受中断允许状态 INTE 的控

制。INTE 的状态可利用 C 口位控操作的置位/复位来控制。例如,用位控操作使 $PC_4 = 1$, 则 A 口的 $INTE_A$ 为 1, 允许中断; 使 $PC_4 = 0$, 则禁止中断。B 口的 $INTE_B$ 是由 PC_2 控制的。

⑤ 除用于 A、B 口输入/输出控制的 6 根线外, C 口剩下的两根线 PC_6 和 PC_7 , 它们的用法与方式 1 输出时 PC_4 、 PC_5 的用法类似。

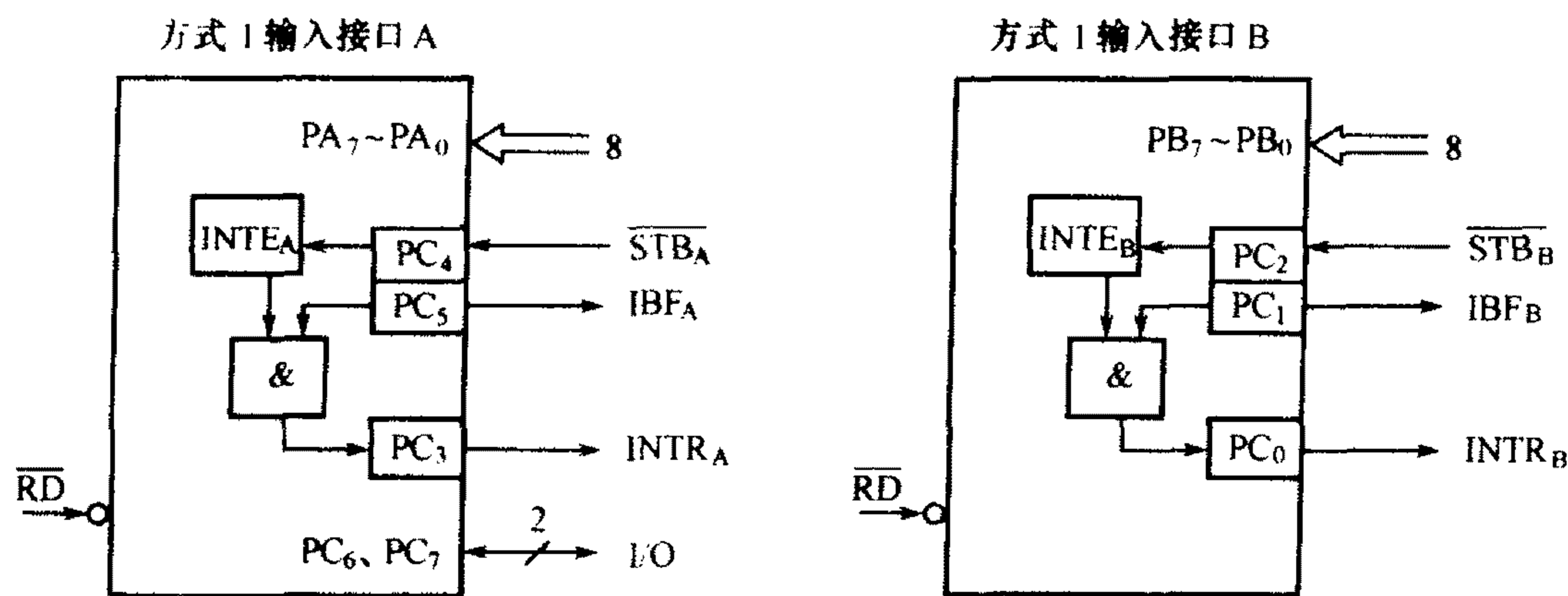


图 8.28 方式 1 下 A、B 口都设定为输入时的控制信号定义

方式 1 下数据输入过程为：当外设要发送数据时, 就将数据送到 8255 的 A 口或 B 口上, 并利用 \overline{STB} 脉冲将数据锁存到 8255 的输入数据锁存器中。 \overline{STB} 还会使 IBF 有效并产生 INTR 信号, 有效的 IBF 通知外设数据已被锁存, INTR 信号则可用于通过 8259 中断控制器向 CPU 提出中断请求, 要求 CPU 从 8255 的输入端口上读取数据。CPU 响应中断并读取数据后使 IBF 和 INTR 变为无效。上述过程可用图 8.29 给出的时序图予以说明。

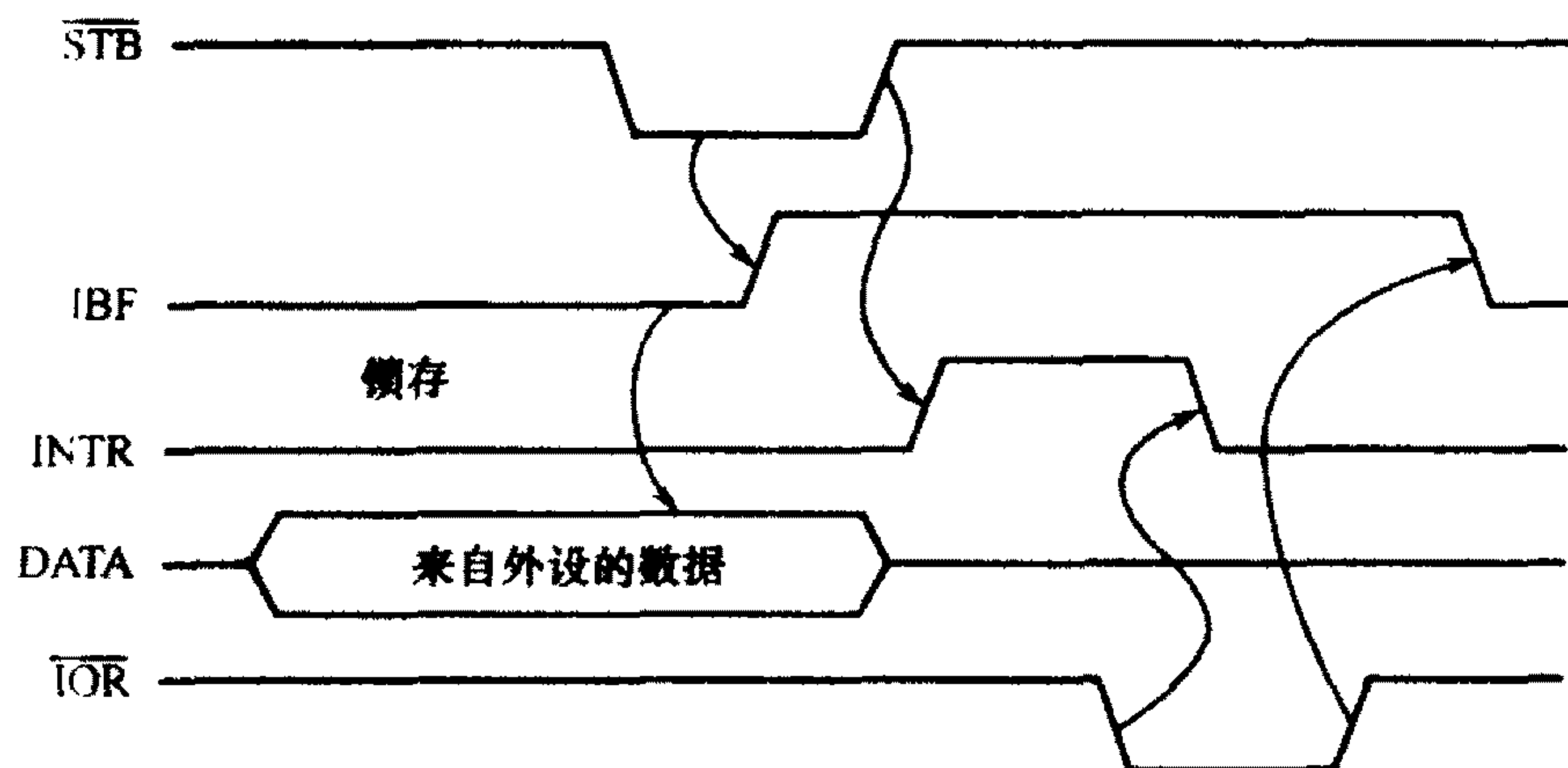


图 8.29 方式 1 下的数据输入时序

以上介绍了 A、B 口同为输出或同为输入的情况, 实际上, 在方式 1 下, 8255 的 A 口和 B 口既可以同时为输入或输出, 也可以一个为输入, 另一个为输出。若 A、B 口的输入/输出方

向不同,则 C 口的控制信号安排仍可参照上述的说明,只是 C 口控制信号的定义应与 A、B 口的输入/输出方向相对应。例如,A 口设置为输入,B 口设置为输出,这时 PC_3 、 PC_4 和 PC_5 分别定义为控制 A 口输入的 \overline{INTR} 、 \overline{STB} 和 \overline{IBF} 信号,而 PC_0 、 PC_1 和 PC_2 分别定义为控制 B 口输出的 \overline{INTR} 、 \overline{OBF} 和 \overline{ACK} 信号。除此之外,还可以设置这两个端口一个工作于方式 1,而另一个工作于方式 0。这种灵活的工作特点是由 8255 可编程的功能决定的。

3) 方式 2

方式 2 又称为双向传输方式。只有 A 口可以工作在这种方式。在双向传输方式下,外设能利用 8 位数据线与 CPU 进行双向通信,即此时 A 口既作为输入口又作为输出口。与方式 1 类似,方式 2 要利用 C 口的 5 根线来提供双向传输所需的控制信号。当 A 口工作于方式 2 时,B 口可以工作在方式 0 或方式 1,而 C 口剩下的 3 根线可作为零散的输入/输出线使用或用做 B 口方式 1 之下的控制信号线。

A 口在工作方式 2 下,C 口各控制信号定义如图 8.30 所示。图中省略了 B 口和 C 口的其他引出线。当 A 口工作于方式 2 时,其控制信号 \overline{OBF} 、 \overline{ACK} 、 \overline{STB} 、 \overline{IBF} 及 \overline{INTR} 的含意与方式 1 时相同。但在时序上有一些不同,主要区别如下:

① 因为在方式 2 下,A 口既作为输出又作为输入,因此只有当 \overline{ACK} 有效时,才能打开 A 口输出数据三态门,使数据由 $PA_0 \sim PA_7$ 输出。当 \overline{ACK} 无效时,A 口的输出数据三态门呈高阻状态。

② A 口输入、输出均有数据的锁存能力。

③ A 口的数据输入或数据输出均可引起中断。从图 8.30 可以看出,输入或输出中断还受到中断允许状态 \overline{INTE}_2 和 \overline{INTE}_1 的影响。 \overline{INTE}_2 是由 PC_4 控制的,而 \overline{INTE}_1 是由 PC_6 控制的。用 C 口的位控操作使 PC_4 或 PC_6 置位或复位,就可以允许或禁止相应的中断请求。

A 口工作于方式 2 的时序如图 8.31 所示。此时的 A 口可以认为是方式 1 的输入和输出的组合(当然,A 口不可能在某一个时刻既输出又输入,而是在某一时刻输出,另一时刻输入。输出/输入操作是分时进行的)。实际传输过程中,输入和输出的顺序以及各自操作的次数是任意的,只要 \overline{IOW} 在 \overline{ACK} 之前发出, \overline{STB} 在 \overline{IOR} 之前发出就可以了。

在输出时,CPU 发出写脉冲 \overline{IOW} ,向 A 口写入数据。 \overline{IOW} 信号使 \overline{INTR} 变低电平,同时使 \overline{OBF} 有效。外设接到 \overline{OBF} 信号后发出 \overline{ACK} 信号,从 A 口读出数据。 \overline{ACK} 信号使 \overline{OBF} 无效,并使

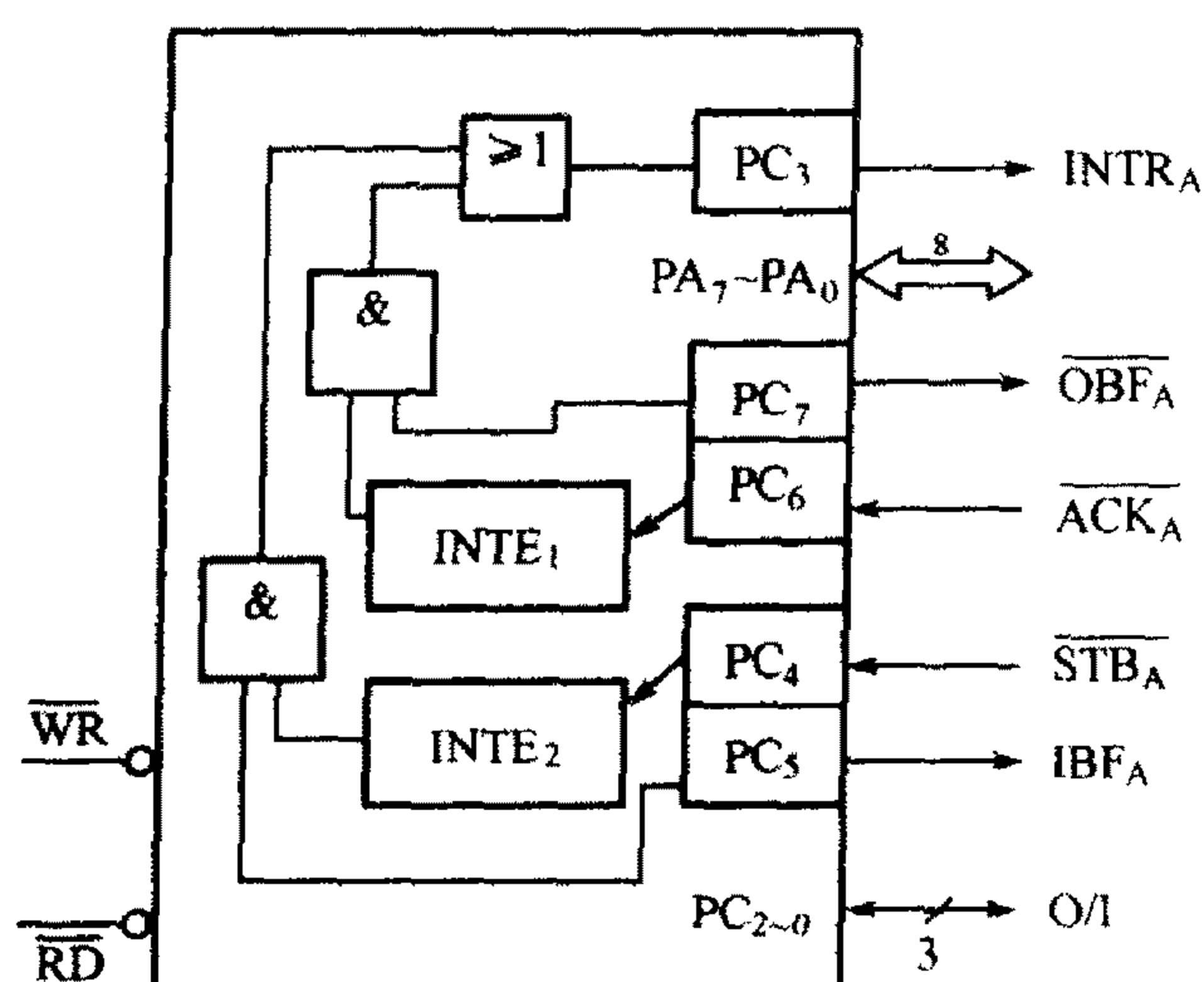


图 8.30 工作方式 2 下的信号定义

INTR 变高电平(如图 8.31 所示的虚线部分),产生中断请求,准备输出下一个数据。

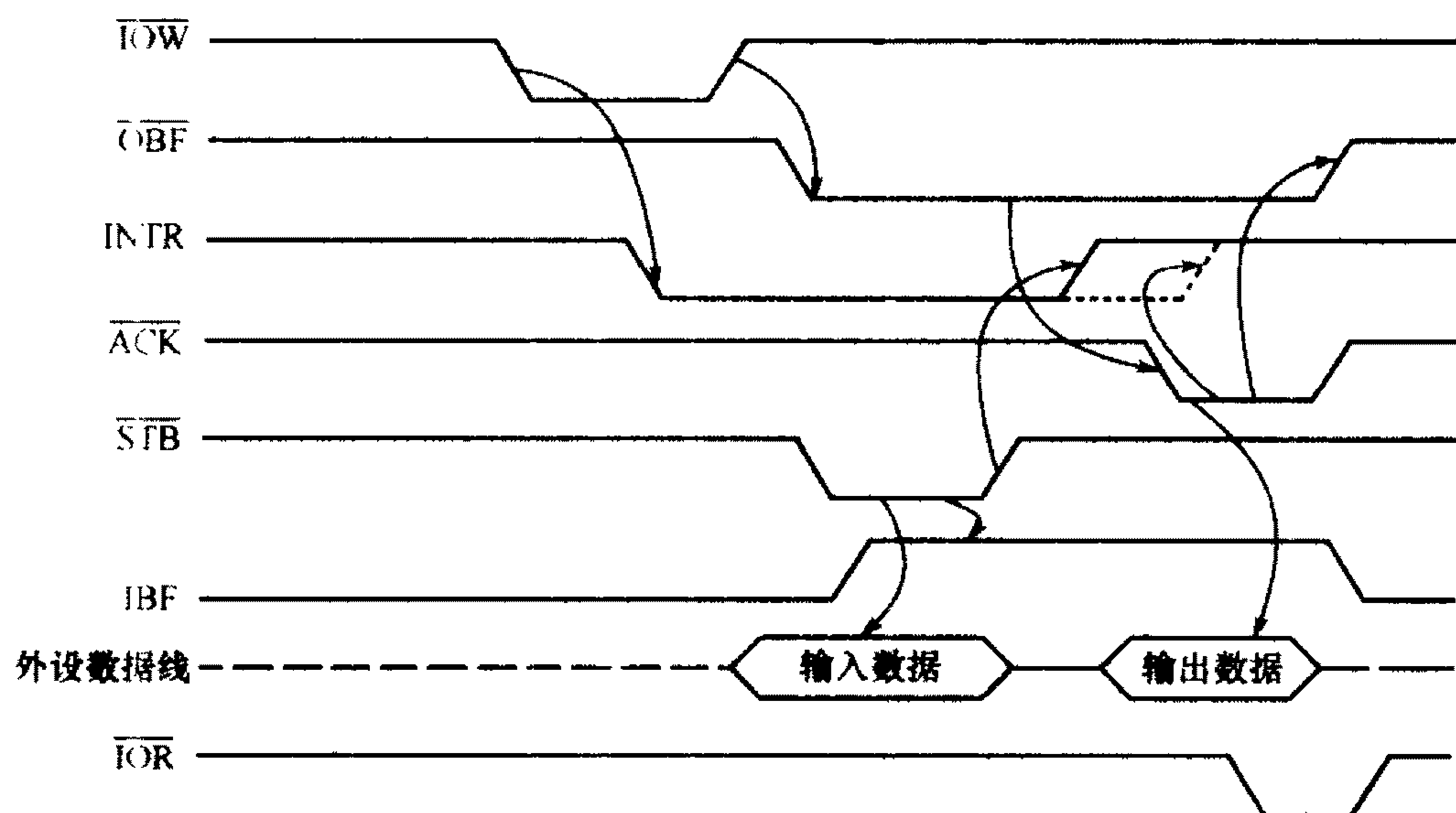


图 8.31 方式 2 下的工作时序

输入时,外设向 8255 送来数据,同时发 \overline{STB} 信号给 8255,该信号将数据锁存到 8255 的 A 口,从而使 IBF 有效。 \overline{STB} 信号结束使 INTR 有效,向 CPU 请求中断。CPU 响应中断后,发出读信号 \overline{IOR} ,从 A 口中将数据读走。 \overline{IOR} 信号会使 INTR 和 IBF 信号无效,从而开始下一个数据的读入过程。

值得注意的是,在工作方式 2 下,8255 与外设之间是通过 A 口的 8 根线 $PA_0 \sim PA_7$ 交换数据的。在 $PA_0 \sim PA_7$ 上,随时可能出现输出到外设的数据,也可能出现外设送给 8255 的数据,这就要防止 CPU 和外设同时竞争 $PA_0 \sim PA_7$ 数据线。

3. 8255 的方式控制字及状态字

由前面的叙述已知,8255 具有三种工作方式,可以利用软件编程来设置 8255 的 3 个端口工作于何种方式。这里所谓的软件编程就是向芯片中的控制寄存器送不同的控制字,从而确定 8255 的工作方式。这个过程也称为 8255 的初始化。在实际应用中,可根据不同的需要,通过初始化使 8255 的 3 个端口工作在不同的方式。

1) 控制字

8255 的控制字由 8 位二进制数组成,各位的定义如图 8.32 所示。

在图 8.32 中,(a)为方式控制字,(b)为 C 口的位控制字。由图可知,控制字第 7 位的状态决定当前的控制字是方式控制字还是 C 口的位控字。

当 $b_7 = 1$ 时,该控制字为方式控制字,用于确定各端口的工作状态。其中的 $b_6 \sim b_3$ 用来控制 A 组,即 A 口的 8 位和 C 口的高 4 位。而控制字的低 3 位 $b_2 \sim b_0$ 用来控制 B 组,包括 B

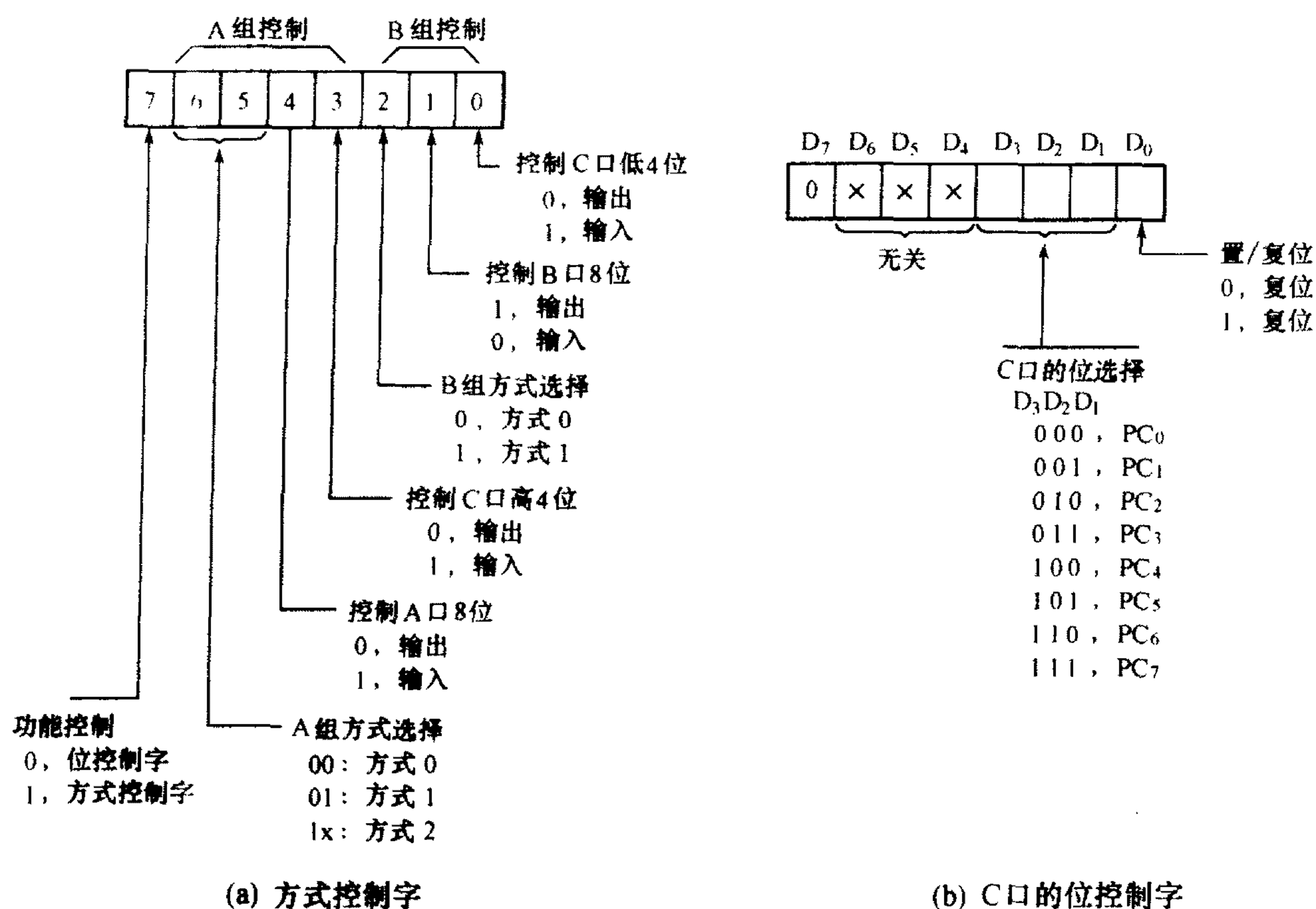


图 8.32 8255 的控制字格式

口的 8 位和 C 口的低 4 位。

当 $b_7 = 0$ 时, 该控制字为 C 口的位控字——对指定位进行置位或复位。利用 C 口位控制字, 可使 C 口的某一位输出 0 或输出 1。

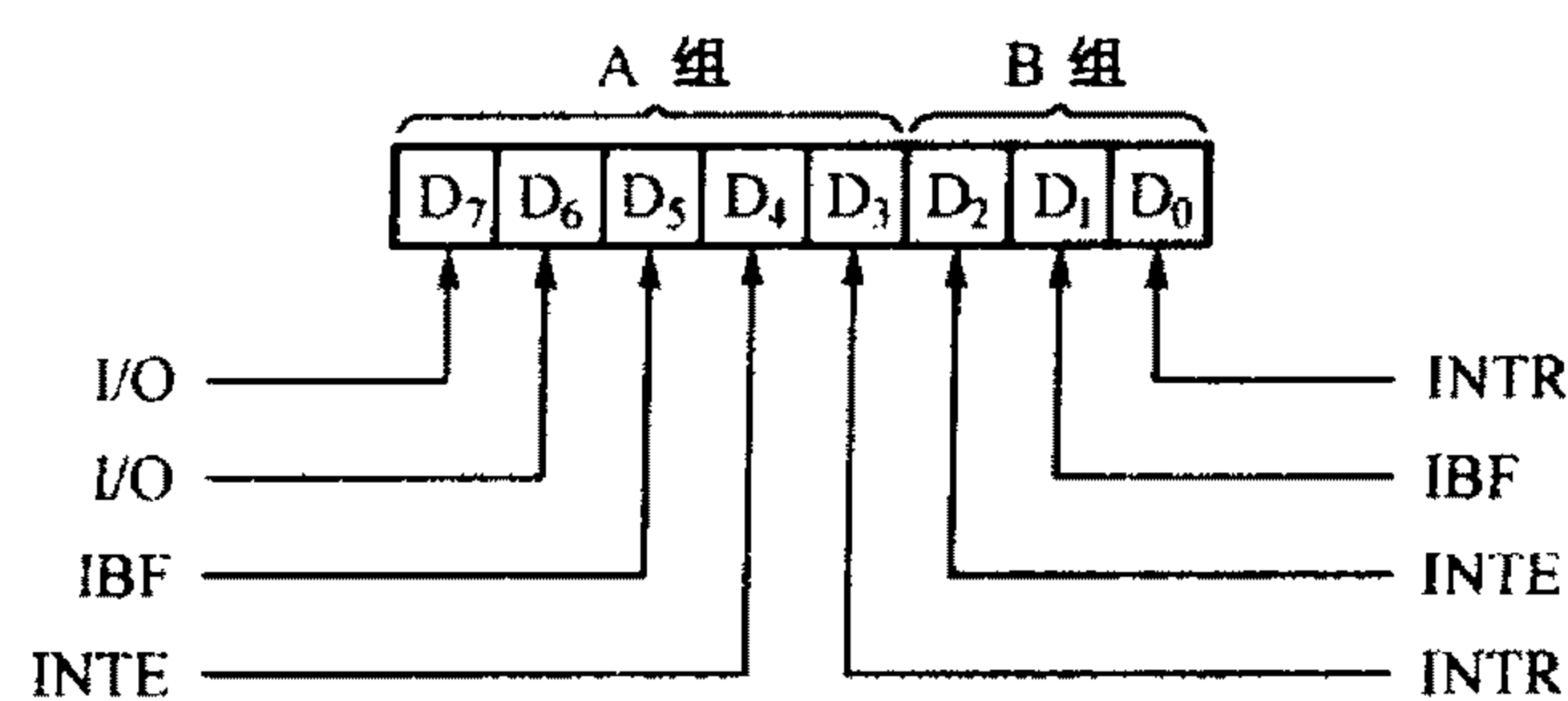
2) 状态字

状态字反映了 C 端口各位当前的状态。当 8255 的 A 口、B 口工作在方式 1 或 A 口工作在方式 2 时, 通过读 C 口的状态, 可以检测 A 口和 B 口当前的工作情况。A、B 口工作在不同方式下的状态字各位的含意分别如图 8.33(a)、(b) 和 (c) 所示。其中低 3 位 $D_0 \sim D_2$ 由 B 口的工作方式来决定。当为方式 1 输入时, 其定义如图 8.28 所示。当工作在方式 1 输出时, 与图 8.26 所定义的 $D_0 \sim D_2$ 相同。

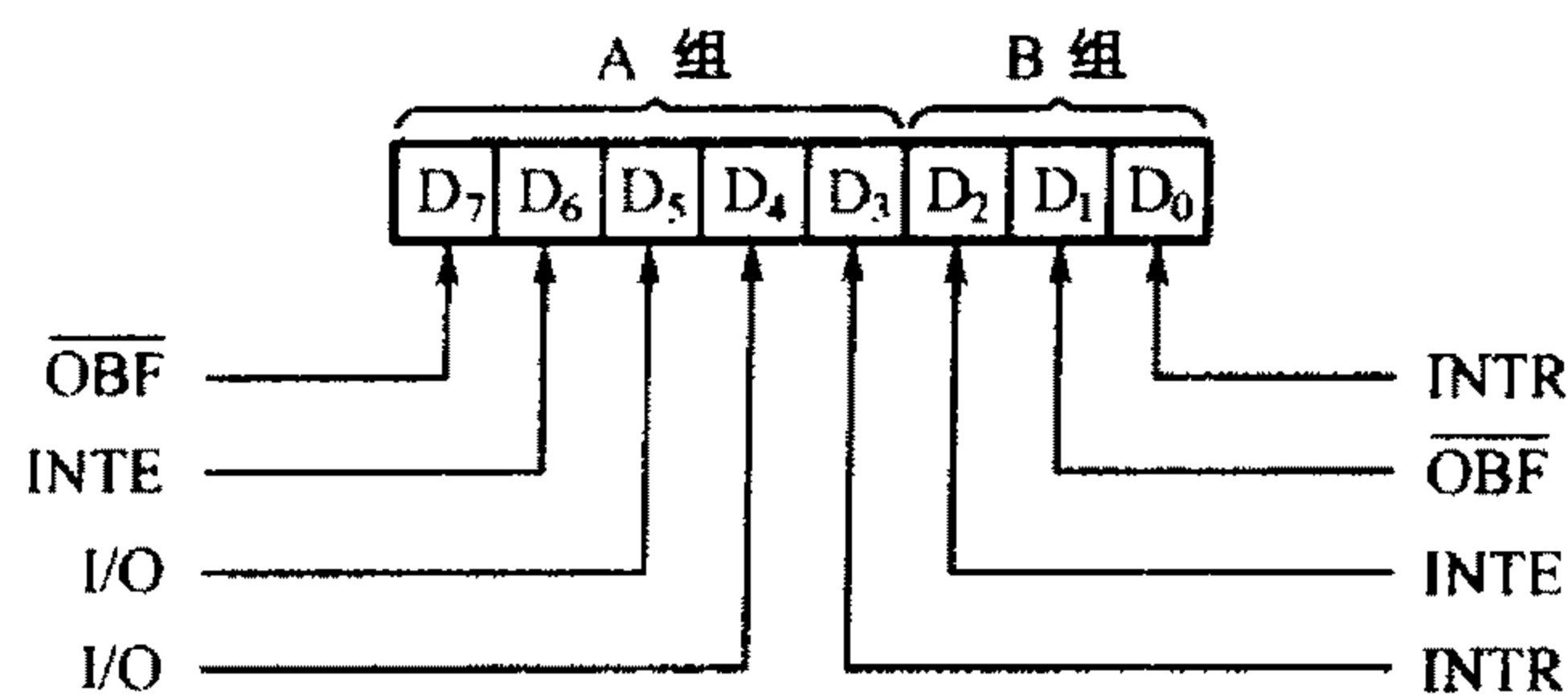
需要说明的是, 图 8.33(a) 和 (b) 分别表示在方式 1 下, A 口、B 口同为输入或同为输出的情况。若在此方式下, A 口、B 口的输入/输出方式不同时, 状态字为上述两状态字的组合。

4. 8255 的应用

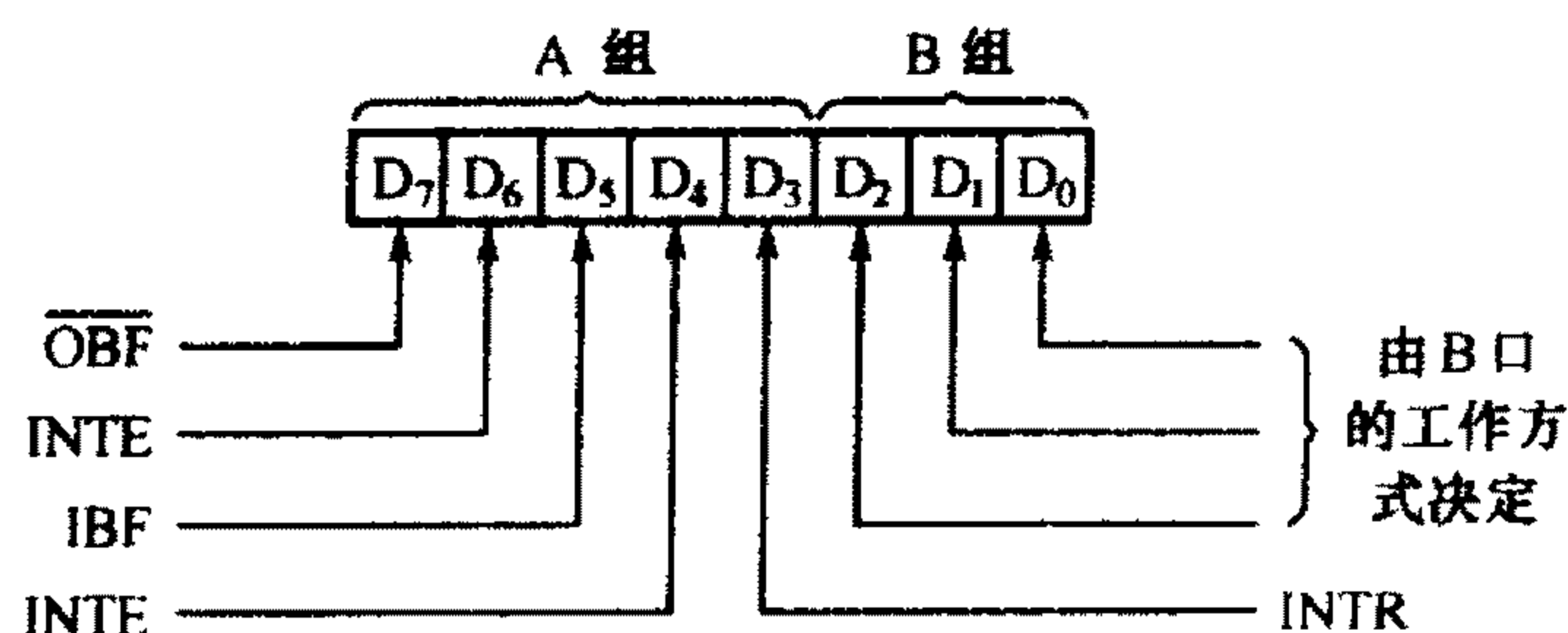
1) 8255 与系统的连接



(a) A、B 口均为方式 1 输入时的状态字



(b) A、B 口均为方式 1 输出时的状态字



(c) A 口工作于方式 2 时的状态字

图 8.33 8255 的状态字

8255 内部包括 A、B、C 3 个端口和一个控制寄存器,共占 4 个外设地址。由高位地址线通过译码产生的片选信号和 A_0 、 A_1 来决定,例如当 $A_0A_1 = 00$ 时指向的是 A 口(参见表 8-5)。对每一个端口都可以分别进行读/写操作。例如,读 A 口是 CPU 将 A 口的数据读入到 AL 寄存器;写 A 口是 CPU 将 AL 中的数据写入 A 口输出。对这 4 个地址进行不同操作时各引脚的状态见表 8-6。根据表 8-6,可以很方便地实现 8255 与系统总线的连接。图 8.34 所示的是利用全译码方式将一片 8255 连接到系统总线上。图中芯片所占的地址范围由 $A_{15} \sim A_2$ 决定,为 $0FF00H \sim 0FF03H$ 。而 A_0 和 A_1 的状态则决定寻址芯片的哪一个端口或控制寄存器。

表 8-6 8255 控制信号状态定义

\overline{CS}	A_1	A_2	\overline{IOR}	\overline{IOW}	操 作
0	0	0	0	1	读 A 口
0	0	1	0	1	读 B 口
0	1	0	0	1	读 C 口
0	0	0	1	0	写 A 口
0	0	1	1	0	写 B 口
0	1	0	1	0	写 C 口
0	1	1	1	0	写控制寄存器
1	x	x	1	1	$D_0 \sim D_7$ 三态

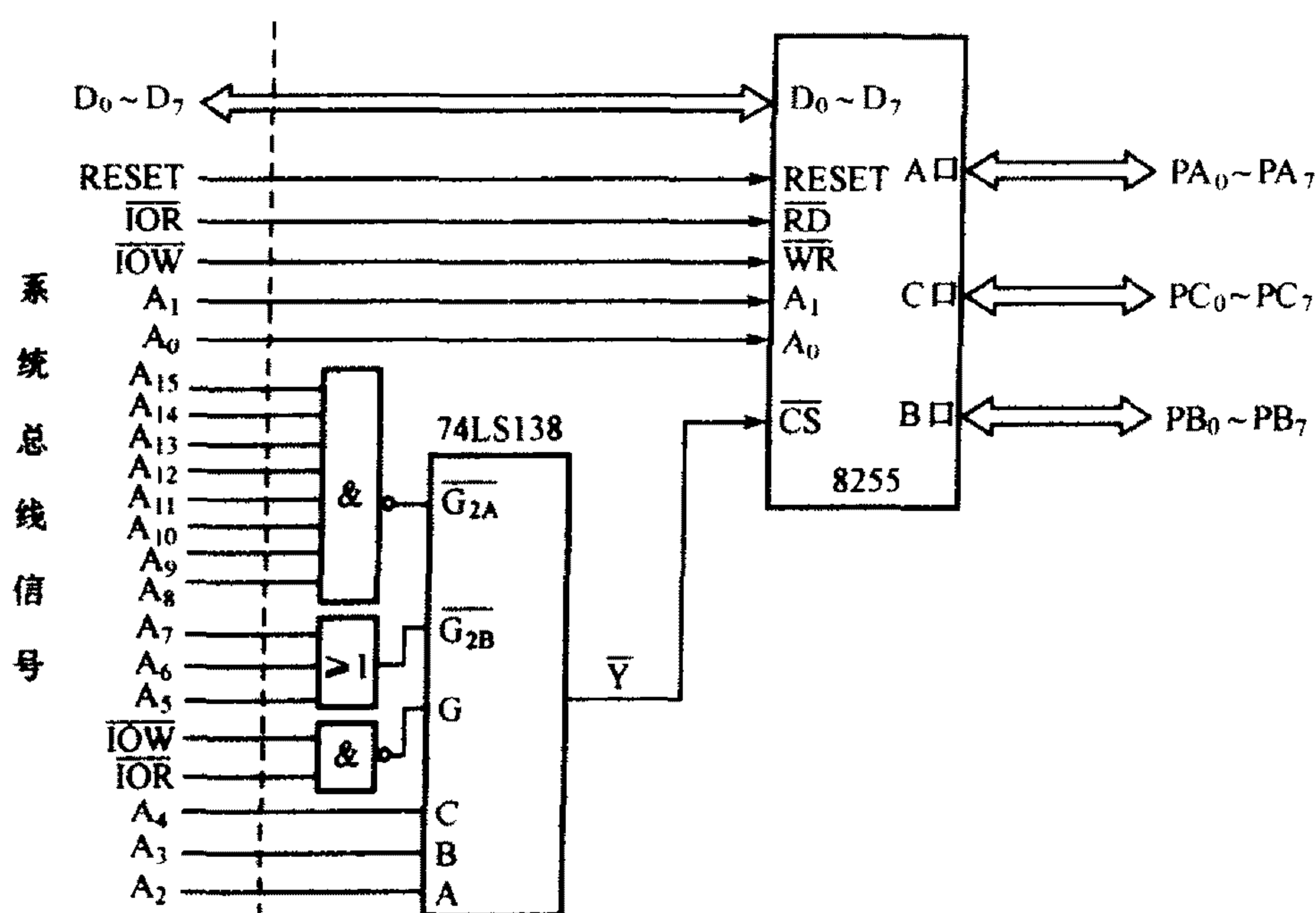


图 8.34 用全译码方式把 8255 连接到系统总线

2) 初始化及应用举例

由上面的叙述可知,8255 在使用之前必须进行初始化,即将适当的控制字写入 8255 的控制寄存器中。在数据传送过程中,CPU 还要通过 8255 向外设发出控制信号并接收外设的状态信息。下面给出几个 8255 的初始化和实际应用的程序例子。

(1) 用查询方式工作的打印机接口

利用 8255 作为打印机的接口,并通过该打印机接口打印字符串,字符串长度放在数据段的 COUNT 单元中,要打印的字符存放在从 DATA 单元开始的数据区中。

8255 与系统及打印机的连接如图 8.35 所示。图 8.36 是打印机的工作时序图。CPU 通

过 8255 接口将数据传送到打印机的 $D_0 \sim D_7$ 端,然后利用一个负脉冲 $\overline{\text{STROBE}}$ (宽度 $\geq 1 \mu\text{s}$) 将数据锁存在打印机内部,以便打印机进行处理。同时,打印机的 BUSY 端送出高电平信号,表示其正忙。仅当 BUSY 端信号变低电平后,CPU 才可以将下一个数据送给打印机。

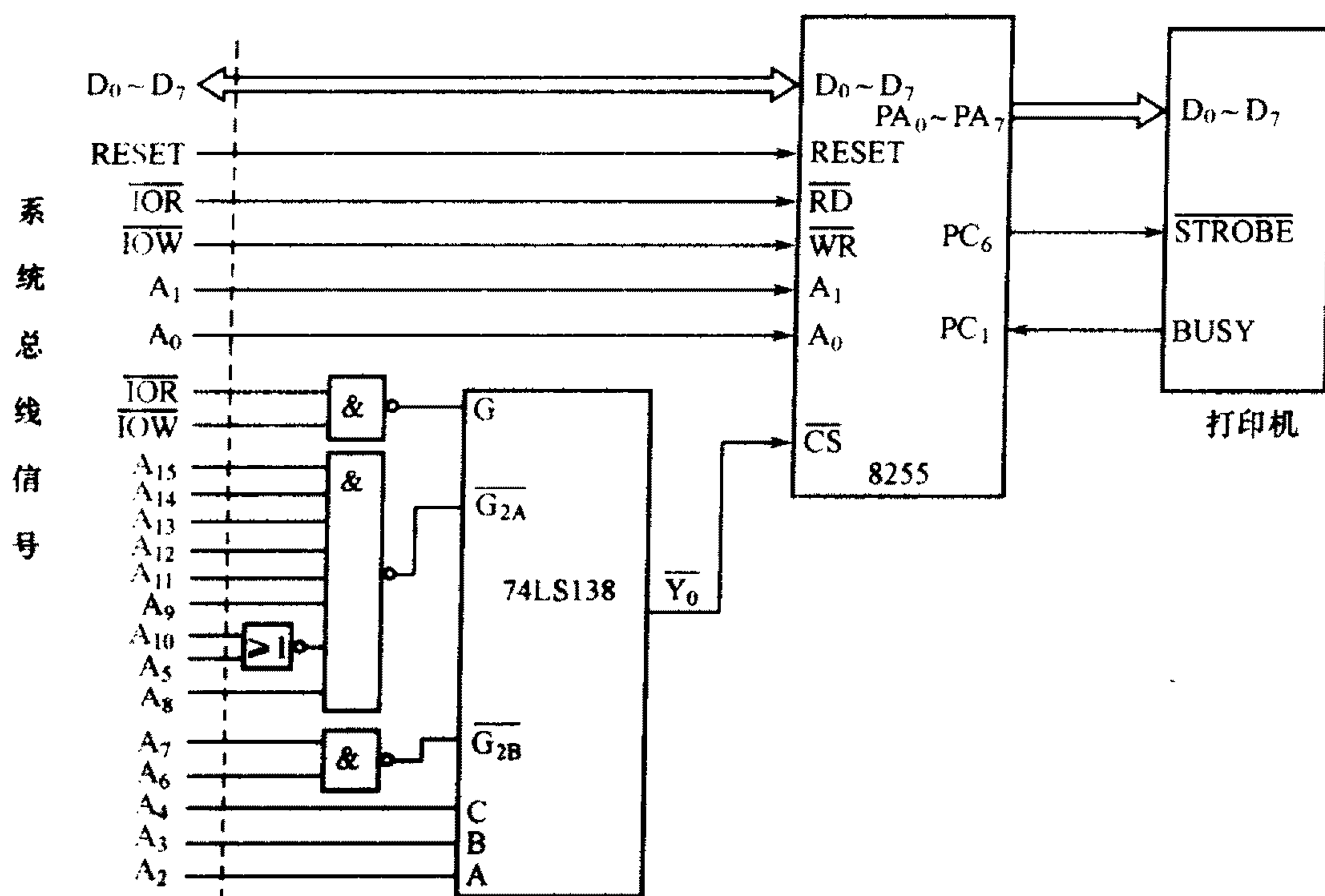


图 8.35 8255 与打印机的连接

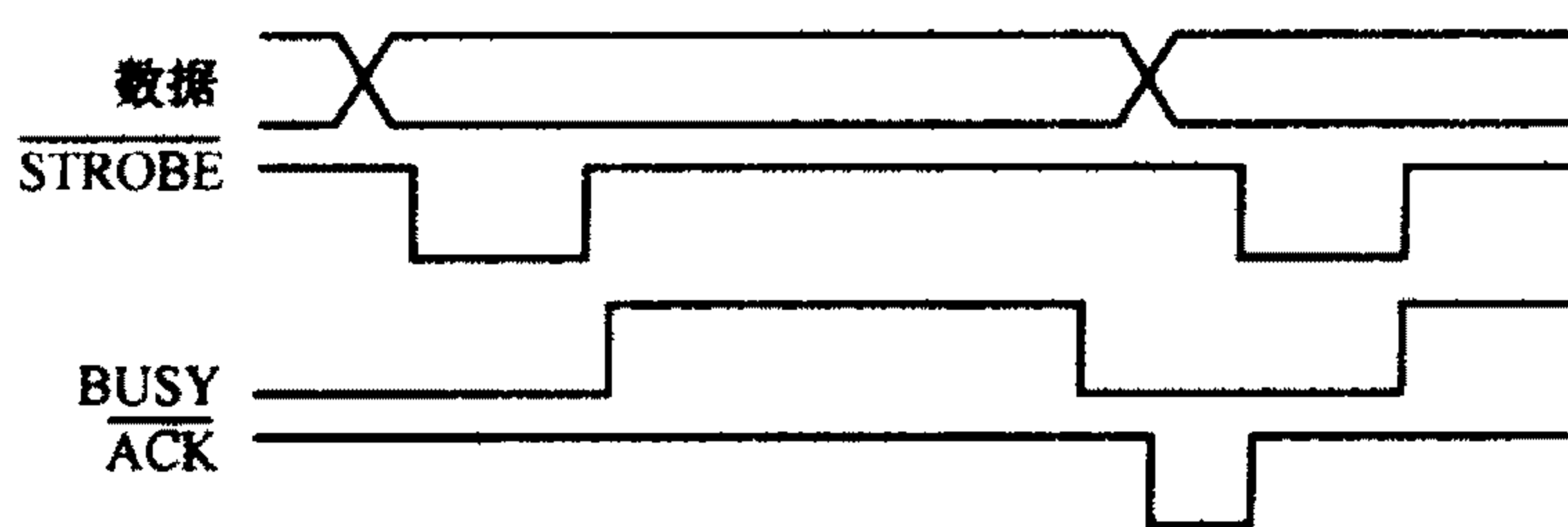


图 8.36 打印机工作时序图

根据上述需求,本例中对 8255 的 3 个口均设置为工作方式 0:

① A 口设置为输出口,向打印机输出数据。

② C 口的 PC_6 用做选通输出,与 $\overline{\text{STROBE}}$ 连接; PC_1 用做状态输入,与打印机的忙信号 (BUSY) 连接。因此,可通过初始化使 C 口的高 4 位为输出,C 口的低 4 位为输入。另外,由于数据输出后要通过 PC_6 端输出一个负脉冲,故在初始化时先要将 PC_6 初始化为高电平。

③ B 口不使用,初始化时可任意定义为输入或输出(本例中将 B 口定义为输出口)。

由此可得出 8255 的初始化程序如下：(8255 的地址范围为 0FBC0H ~ 0FBC3H)

```
INIT:  MOV DX,0FBC3H      ;8255 的控制寄存器端口地址送 DX
        MOV AL,10000001B  ;A 组方式 0: A 口输出,C 口高 4 位输出
                                ;B 组方式 0: B 口输出,C 口低 4 位输入
        OUT DX,AL         ;方式控制字送控制寄存器
        MOV AL,00001101B  ;C 口的按位操作控制字,使 PC6 初始状态置为 1
        OUT DX,AL         ;C 口位操作控制字送控制寄存器
        . . . . .
```

下面是打印一批字符的程序段：

```
        MOV CX,COUNT      ;将字符串长度作为循环次数
        MOV SI,OFFSET DATA ;取字符串首地址
GOON:  MOV DX,0FBC2H      ;0FBC2H 为 C 口的地址
        IN AL,DX          ;从 C 口读入打印机的 BUSY 信号状态
        AND AL,02H        ;测试打印机状态(位 1)
        JNZ GOON          ;若 BUSY 为高电平,则循环等待
        MOV AL,[SI]       ;否则取一个字符
        MOV DX,0FBC0H     ;0FBC0H 为 A 口的地址
        OUT DX,AL         ;输出一个字符到 A 口
        MOV DX,0FBC2H     ;准备在 PC6 上生成一个负脉冲
        MOV AL,0
        OUT DX,AL         ;因仅 PC6 接打印机,故由 C 口输出 00H 将使 PC6 变低
        MOV AL,40H
        OUT DX,AL         ;再使 PC6 变高,在 PC6 上生成一个 $\overline{\text{STROBE}}$ 负脉冲
        INC SI            ;指向下一个字符
        LOOP GOON         ;若未结束,则继续
        HLT
```

程序中,在 PC₆ 引脚上生成的 $\overline{\text{STROBE}}$ 负脉冲是通过往 C 口输出数据(先将 PC₆ 初始化为 1,输出一个 0,然后再输出一个 1)而形成的。当然,也可以利用 C 口的位控制字对 PC₆ 进行置位/复位操作来实现。如：

```
        MOV DX,0FBC3H
        MOV AL,00001100B   ;PC6 复位(= 0)
        OUT DX,AL
        MOV AL,00001101B   ;PC6 置位(= 1)
        OUT DX,AL
```

(2) 用中断方式工作的打印机接口

要求同上例,但使 8255 工作在方式 1 下,并利用中断方式进行打印。这时 8255 与打印机的电路连接方法如图 8.37 所示。

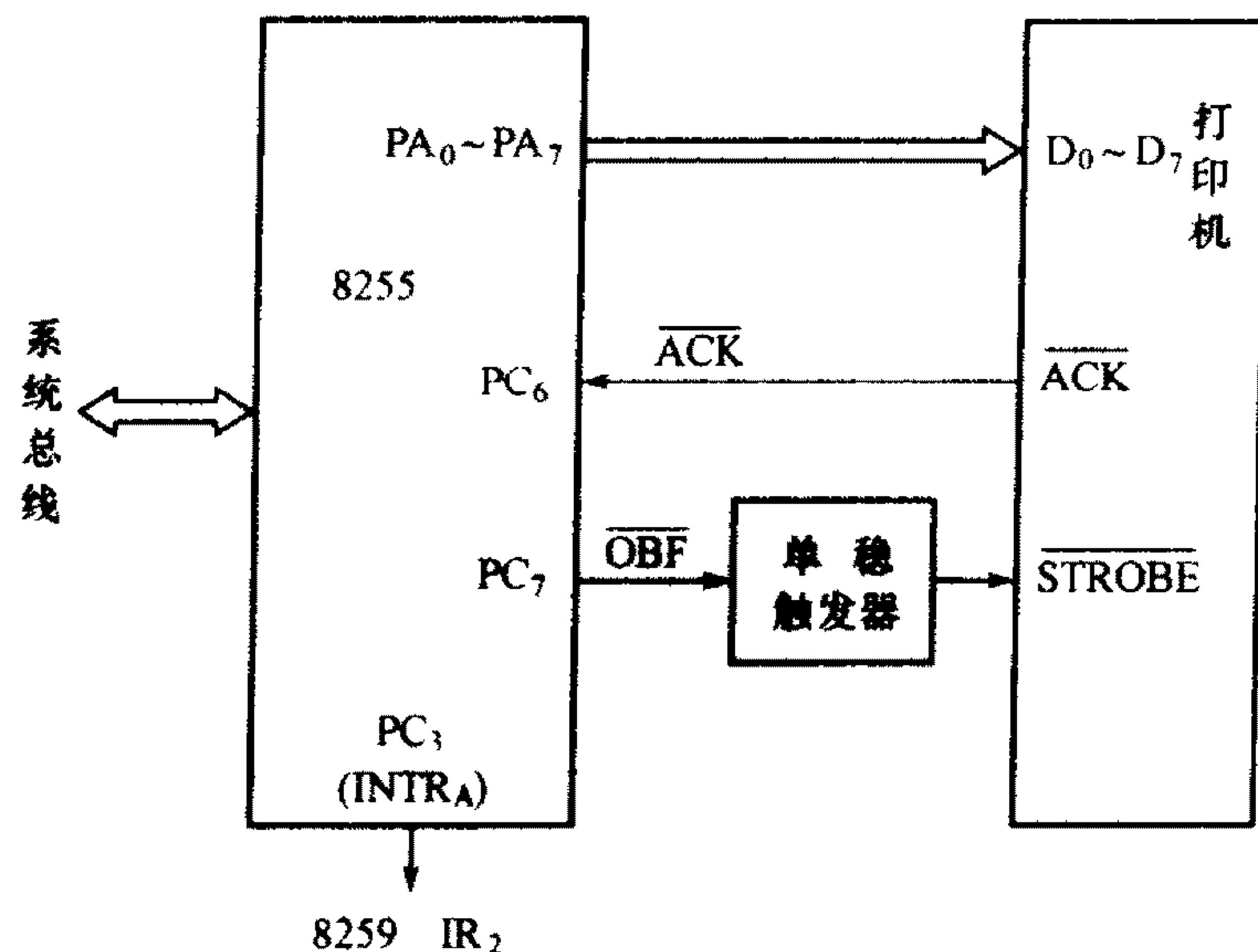


图 8.37 8255 工作于方式 1 下与打印机的连接示意图

从打印机的工作时序图可知,打印机每接收一个字符后,会送出一个低电平的响应信号 $\overline{\text{ACK}}$ 。利用这个信号,可使工作于方式 1 的 8255 产生中断来请求 CPU 发送下一个字符。

将 8255 设置为方式 1, A 口为数据输出口,此时 PC_7 为 $\overline{\text{OBF}}$ 信号的输出端, PC_6 为 $\overline{\text{ACK}}$ 信号的输入端,而 PC_3 为 INTR 信号的输出端,将其接到 8259 的 IR_2 端,所以中断类型为 0AH。

输出时,先输出一个空字符,以引起中断过程。在中断中逐个输出要打印的字符,每次中断 CPU 就输出一个字符。要打印的字符送入 8255 后,会使 $\overline{\text{OBF}}$ 信号变低电平,利用 $\overline{\text{OBF}}$ 的下降沿触发一单稳触发器,产生打印机所需要的 STROBE 脉冲,将字符送给打印机。接收到字符后,打印机发出 $\overline{\text{ACK}}$,清除 $\overline{\text{OBF}}$ 标志,并产生有效的 INTR 输出,形成新的中断请求, CPU 响应中断后再输出下一个字符。

8255 的端口地址仍为 0FBC0H ~ 0FBC3H。为简单起见,在初始化 8255 时,仍使 B 口工作于方式 0 输出, C 口的其余 5 根线均定义为输出,故控制字为 10100000B,即 0A0H。

要使 PC_3 能够产生中断请求信号 INTR,还必须使 A 口的中断请求允许状态 $\text{INTE} = 1$ 。这是通过 8255 的置位/复位操作将 PC_6 置 1 来实现的,即在初始化 8255 时除了写方式控制字外,还要写 C 口的位控制字。

图 8.38 所示是采用中断方式向打印机输出字符的程序流程图,包括主程序和中断处理子程序两部分。主程序完成以下 3 项工作:将中断服务子程序的入口地址送入中断向量表、

开中断等待中断的准备工作以及 8255 的初始化。而中断服务子程序则完成字符的输出。

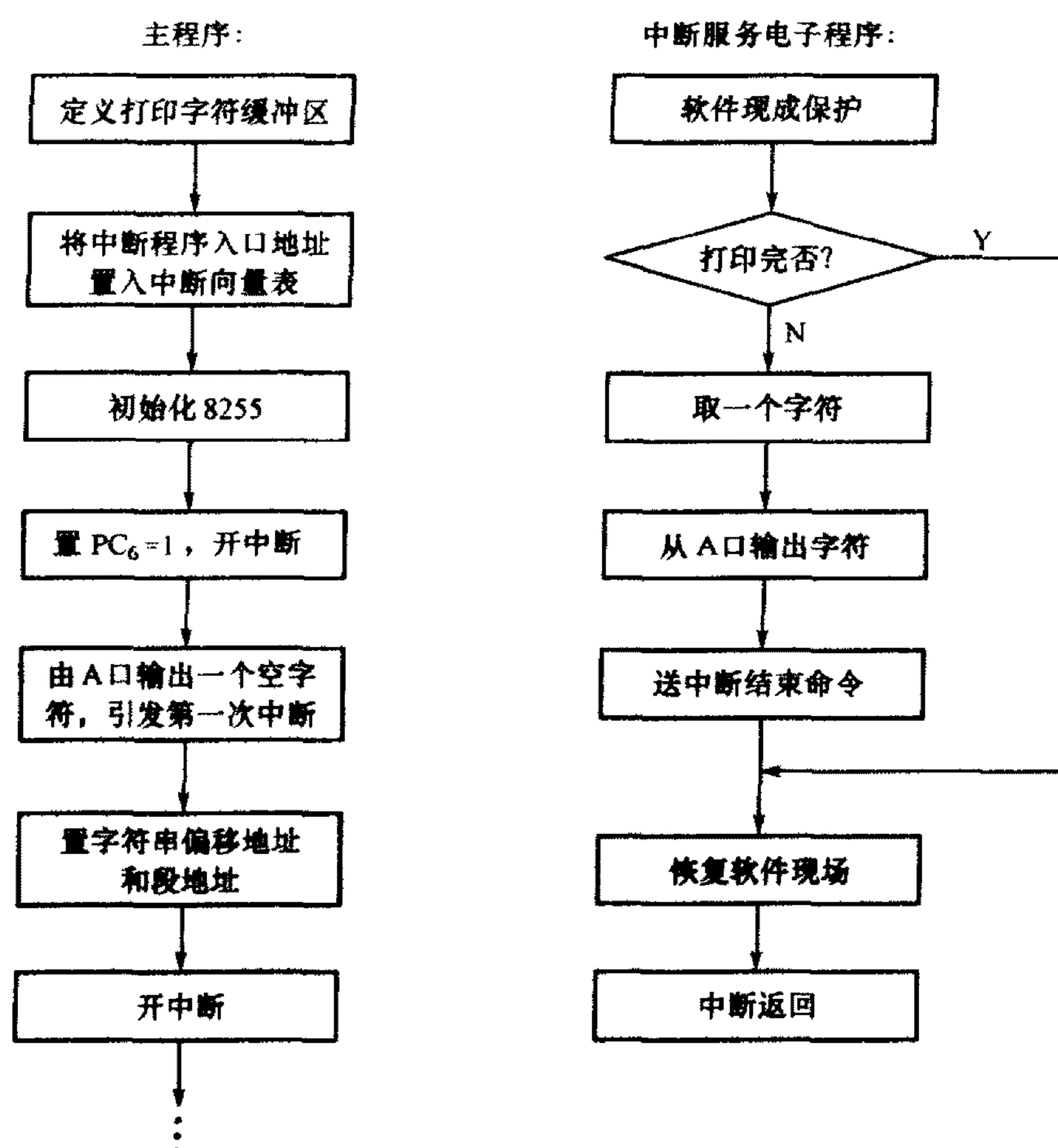


图 8.38 8255 采用中断方式向打印机输出字符的程序流程

8.2.3 可编程串行接口 8250

Intel 8250 是专用于异步串行通信的可编程串行接口芯片,具有很强的串行通信能力和灵活的可编程功能,在微型计算机中应用极为广泛。

1. 引脚及功能

可编程串行通信接口 8250 的外部引脚图如图 8.39 所示,共有 40 根引脚,单电源 +5 V 供电。除电源线(V_{CC})和地线(GND)外,其引出线信号可分为面向系统和面向通信设备两大类。下面分别介绍它们的意义及功能。

1) 面向系统的引出线信号

$D_0 \sim D_7$ 双向数据线。与系统数据总线相连接,用以传送数据、控制信息和状态信息。

CS₀、CS₁、 $\overline{\text{CS}}_2$ 片选信号,输入。只有当它们同时有效,即 CS₀ = 1, CS₁ = 1, $\overline{\text{CS}}_2$ = 0 时,才能选中该 8250 芯片。

CSOUT 片选输出信号。当 8250 的 CS₀、CS₁ 和 $\overline{\text{CS}}_2$ 同时有效时,CSOUT 为高电平。

A₀ ~ A₂ 8250 内部寄存器的选择信号。它们的不同编码,可以选中 8250 内部不同的寄存器。

ADS 地址选通信号,低电平有效。有效时可将 CS₀、CS₁、 $\overline{\text{CS}}_2$ 及 A₀、A₁、A₂ 锁存于 8250 内部。若在工作中不需要锁存上述信号,则可将 ADS 直接接地,使其恒有效。

DISTR、 $\overline{\text{DISTR}}$ 数据输入选通信号。当它们其中任何一个有效时(DISTR 为高电平或 $\overline{\text{DISTR}}$ 为低电平),被选中的 8250 寄存器内容可被读出。它们经常与系统总线上的 $\overline{\text{IOR}}$ 信号相连接。当它们同时无效时,8250 不能读出。

DOSTR、 $\overline{\text{DOSTR}}$ 数据输出选通信号。当它们其中一个有效时(DOSTR 为高电平或 $\overline{\text{DOSTR}}$ 为低电平),被选中的 8250 寄存器可写入数据或控制字。它们常与系统总线的 $\overline{\text{IOW}}$ 相连。当它们同时无效时,8250 则不能写入。

DDIS 驱动器禁止信号。该输出信号在 CPU 读 8250 时为低电平,非读时为高电平。可用此信号来控制 8250 与系统总线间的数据总线驱动器。

INTR 中断请求输出信号,高电平有效。当 8250 中断允许时,接收出错、接收数据缓冲器满、发送数据保持器空以及 MODEM 的状态均能够产生有效的 INTR 信号。主复位信号(MR)可使该输出信号无效。

MR 主复位输入信号,高电平有效。通常与系统复位信号 RESET 相连。主复位时,除了接收数据缓冲寄存器、发送数据保持寄存器和除数锁存器外,其他内部寄存器及信号均受到主复位的影响。其功能定义见表 8-7。

2) 面向外部通信设备的引脚信号

SIN 串行数据输入端。外设或其他系统传送来的串行数据由该端进入 8250。

SOUT 串行数据输出端。主复位信号可使其变为高电平。

$\overline{\text{CTS}}$ 清除发送信号,输入,低电平有效。当它有效时,表示提供 $\overline{\text{CTS}}$ 信号的设备可以接收 8250 发送的数据,它是提供 $\overline{\text{CTS}}$ 信号的设备向 8250 发出的 $\overline{\text{RTS}}$ 信号的应答信号。

$\overline{\text{RTS}}$ 请求发送信号,输出,低电平有效。它是 8250 向外设发出的发送数据请求信号。它与下面的信号具有同样的功能。

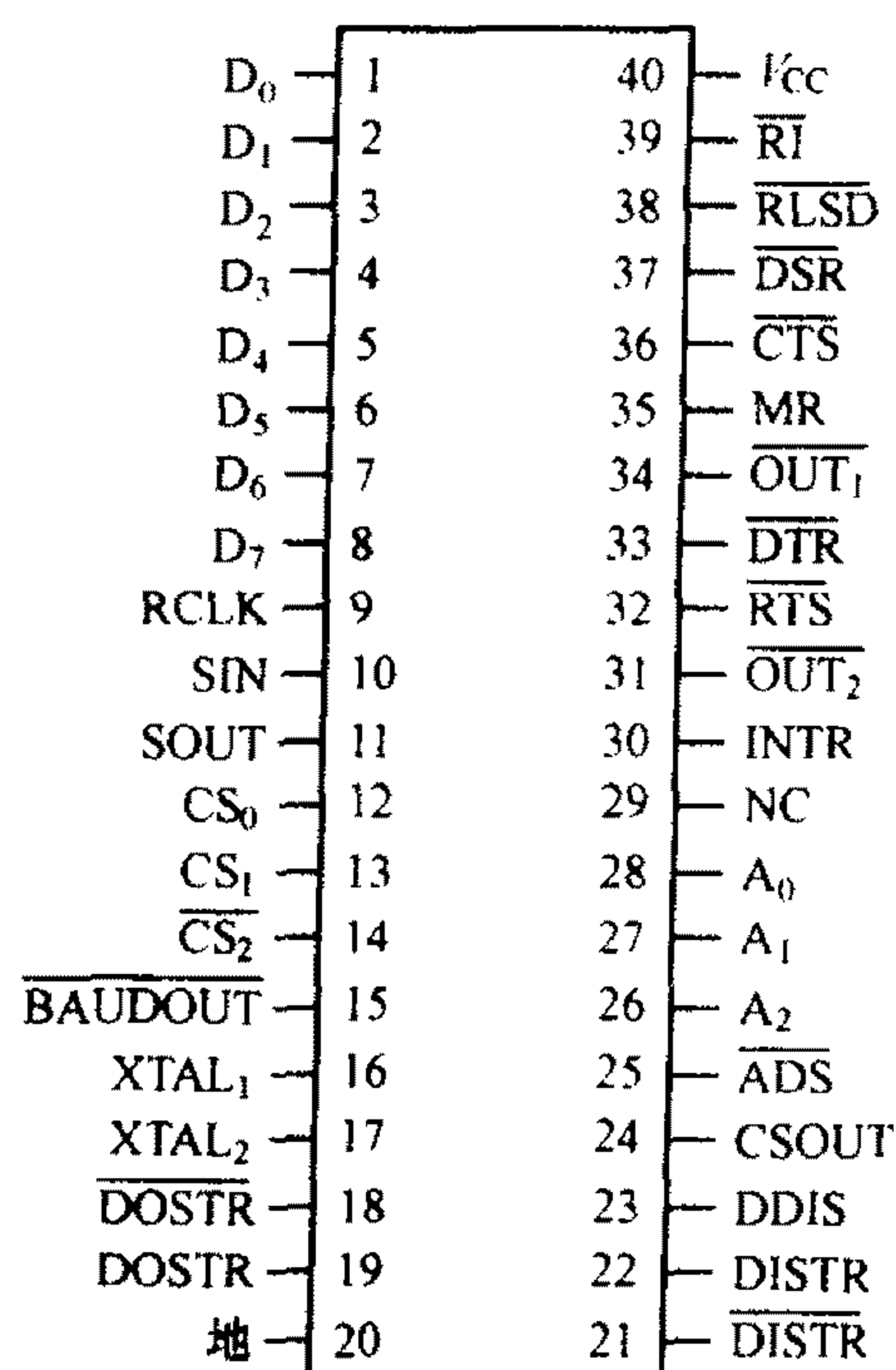


图 8.39 8250 的引脚图

$\overline{\text{DTR}}$ 数据终端准备好信号,输出,低电平有效。它表示 8250 已准备好接收数据。

$\overline{\text{DSR}}$ 数据装置准备好信号,输入,低电平有效。表示接收数据的外设已准备好接收数据,它是对 $\overline{\text{DTR}}$ 信号的应答。

表 8-7 主复位 MR 功能

寄存器或信号	复位控制	复位后的状态
通信控制寄存器	MR	各位均为低电平
中断允许寄存器	MR	各位均为低电平
中断标识寄存器	MR	第 0 位高电平,其余均为低电平
MODEM 控制寄存器	MR	各位均为低电平
通信状态寄存器	MR	除第 5、6 位外其余均为高电平
INTR(线路状态错)	读通信状态寄存器或 MR	低电平
INTR(发送寄存器空)	读中断标志寄存器,写发送数据寄存器或 MR	低电平
INTR(接收寄存器满)	读接收数据寄存器或 MR	低电平
INTR(MODEM 状态改变)	读 MODEM 状态寄存器或 MR	低电平
SOUT	MR	高电平
OUT ₁ 、OUT ₂ 、RTS、DTR	MR	高电平

$\overline{\text{RLSD}}$ 接收线路信号检测信号,输入,低电平有效。表示 MODEM 已检测到载波信号。

$\overline{\text{RI}}$ 振铃指示信号,输入,低电平有效。表示 MODEM 已接收到一个电话振铃信号。

$\overline{\text{OUT}}_1$ 可由用户编程确定其状态的输出端。若用户在 MODEM 控制寄存器第二位 (OUT₁) 写入 1,则 $\overline{\text{OUT}}_1$ 输出端变为低电平。主复位信号 (MR) 可将 $\overline{\text{OUT}}_1$ 置为高电平。

$\overline{\text{OUT}}_2$ 与 $\overline{\text{OUT}}_1$ 一样,也可由用户编程指定。只是要将 MODEM 控制寄存器的第三位 (OUT₂) 写入 1,就可使 $\overline{\text{OUT}}_2$ 变为低电平。主复位信号 (MR) 可将其置为高电平。

$\overline{\text{BAUDOUT}}$ 波特率信号输出。该端输出的是主参考时钟频率除以 8250 内部除数寄存器中的除数后所得到的频率信号。这个频率信号就是 8250 的发送时钟信号,是发送数据波特率的 16 倍。若将此信号接到 RCLK 上,又可以同时作为接收时钟使用。

XTAL₁、XTAL₂ 外部时钟端。这两端可接晶振或直接接外部时钟信号。

RCLK 接收时钟信号。该输入信号的频率为接收数据波特率的 16 倍。

2. 8250 的结构及内部寄存器

8250 的内部结构如图 8.40 所示。

由图可知,8250 中除与系统相连的数据缓冲器、读/写控制逻辑外,还包括 10 个寄存器

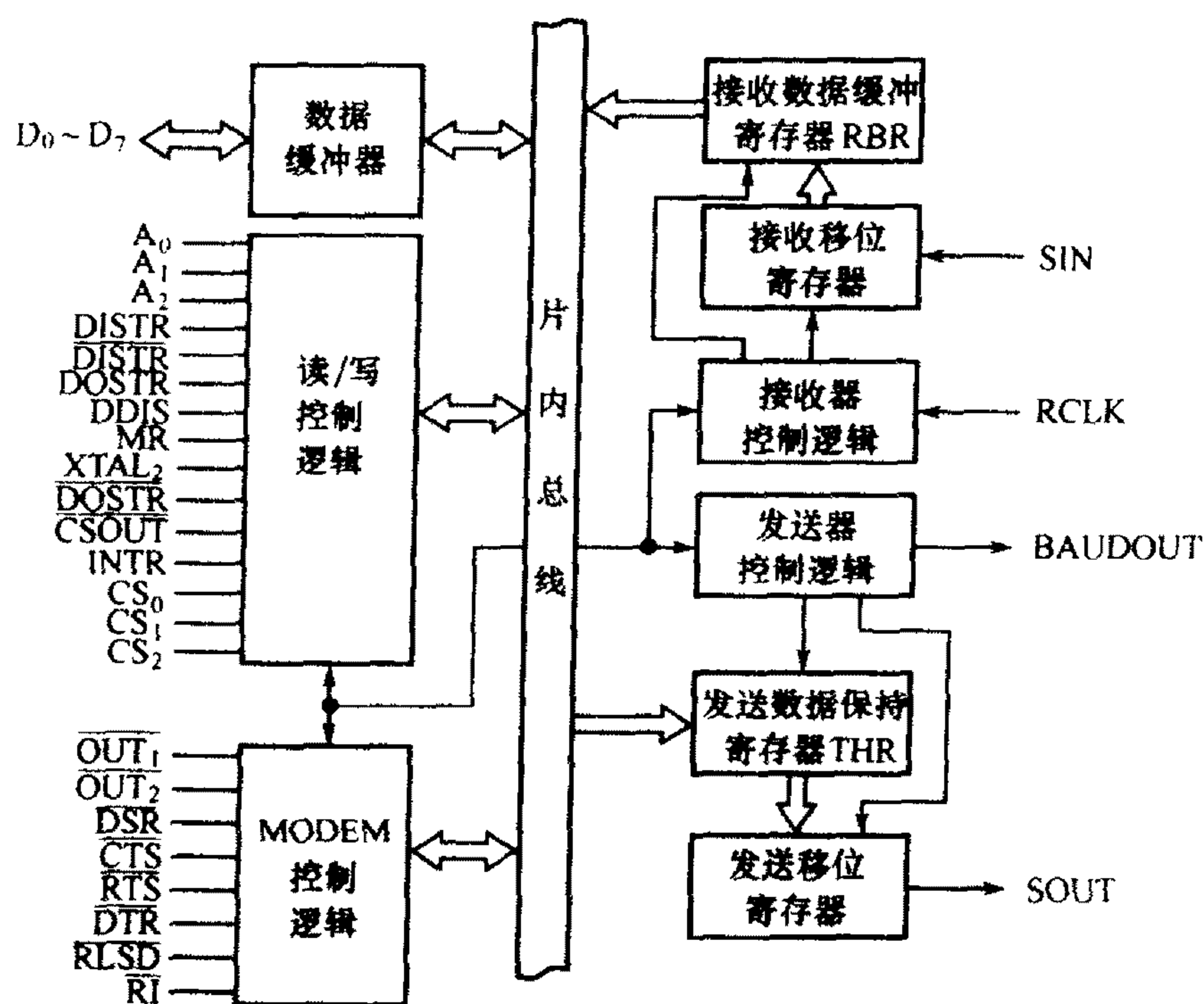


图 8.40 8250 的内部结构框图

(可分为 5 个功能模块)。程序员在对 8250 编程时要经常与这些寄存器打交道,所以要使用 8250 就必须熟练掌握它们各位的意义和使用方法。下面就分别介绍 8250 的这 10 个内部寄存器各位的功能。

1) 通信线路控制寄存器 LCR

这是一个 8 位的寄存器,其各位的主要功能如图 8.41 所示。LCR 主要用于决定在串行通信时所使用的数据格式,例如数据位数、奇偶校验及停止位的多少等。因芯片仅有 3 根地址线,最多只能寻址 8 个寄存器,为此只好使两个除数寄存器和其他寄存器共用地地址。当前是寻址除数寄存器还是其他寄存器,是由 LCR 的最高位 D_7 来区分的。当需要读/写除数寄存器时,必须先使 LCR 的 D_7 置 1,而在读/写其他寄存器时,又必须先将其设为 0。

2) 通信线路状态寄存器 LSR

LSR 是一个 8 位寄存器,其各位的功能如图 8.42 所示。它存放了通信过程中 8250 接收和发送数据的状态。

D_0 为 1 时,表示 8250 已接收到一个完整的字符,CPU 可以从 8250 的接收数据寄存器中读取。一旦读取后,此位即变为 0。

D_1 越限状态标志。接收数据寄存器中的前一数据还未被 CPU 读走,而后一个数据已经到来并将其破坏时,此位为 1。当 CPU 读接收数据寄存器时,使此位变为 0。

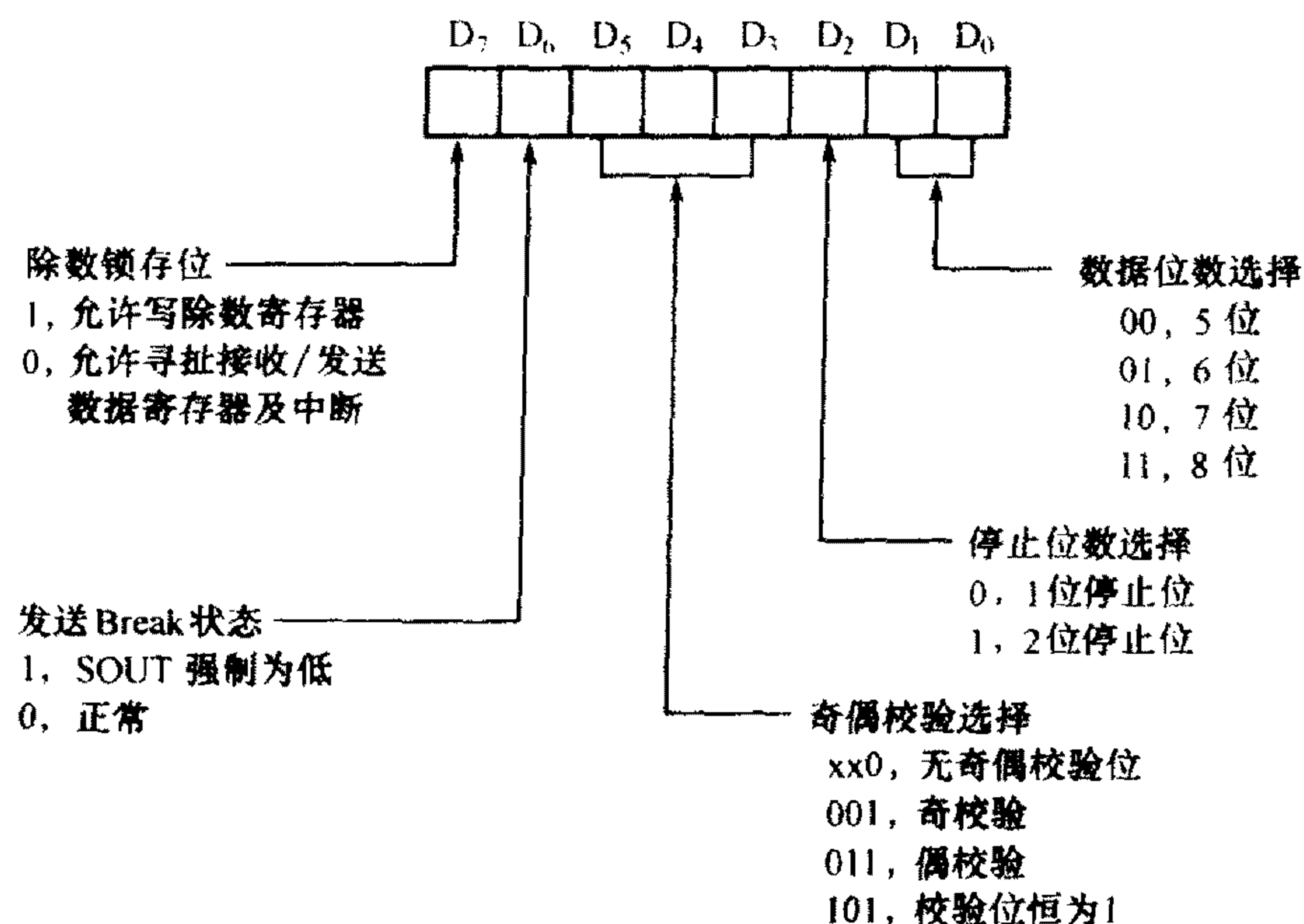


图 8.41 通信线路控制寄存器 LCR

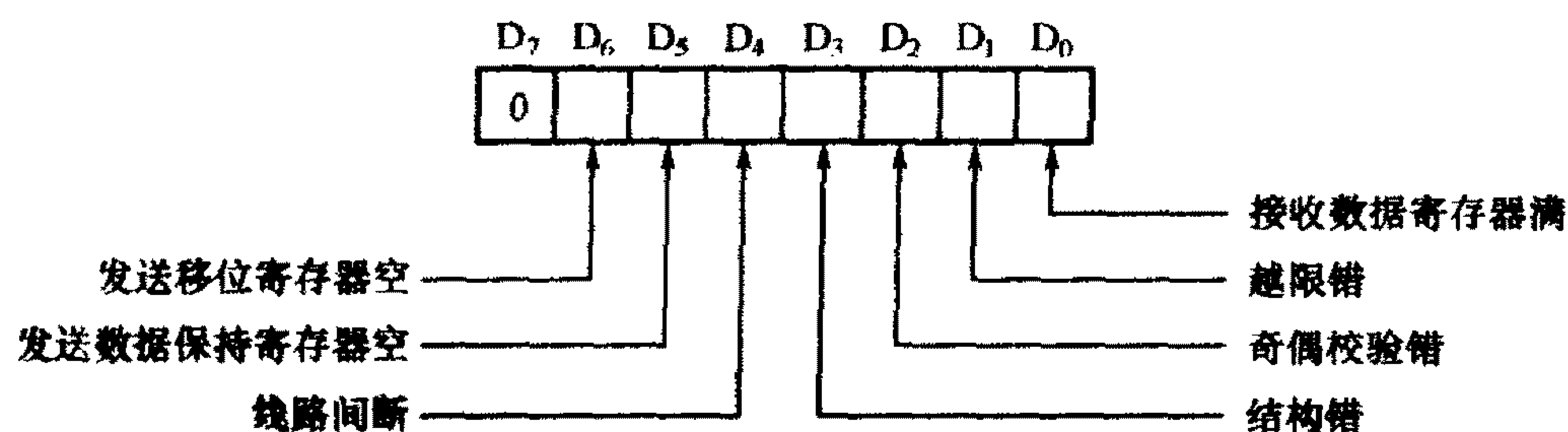


图 8.42 通信线路状态寄存器 LSR

D₂ 奇偶校验错标志。在 8250 对收到的一个完整的字符编码进行奇偶校验时,若发现其值与规定的奇偶校验不同,则使此位为 1,表示数据可能有错。当 CPU 读 LCR 时,此位变为 0。

D₃ 结构错标志。当接收到的数据停止位个数不正确时,此位置 1。当 CPU 读 LCR 时,此位变为 0。

D₄ 线路间断标志。若在一个完整的字符编码的时间间隔中收到的均为空闲状态,则此位置 1,表示线路信号间断。当 CPU 读通信状态寄存器时,使此位变为 0。

出现以上四种状态中的任何一种都会使 8250 发出线路状态出错中断。

D₅ 为 1 时,表示发送数据保持寄存器 THR 空。CPU 将数据写入 THR 后,此位清零。

D₆ 为 1 时,表示发送移位寄存器 TSR 空。当 THR 的数据送入 TSR 时,此位清零。

D_7 恒为 0。

3) 发送数据保持寄存器 THR

这是一个 8 位的寄存器。发送数据时, CPU 将数据写入 THR。只要 TSR 空, THR 中的数据便会由 8250 的硬件自动送入 TSR 中, 以便串行移出。

4) 接收数据缓冲寄存器 RBR

这是一个 8 位的寄存器。8250 接收到一个完整的字符时, 便会把该字符从接收移位寄存器 RSR 传送到 RBR。CPU 可由 RBR 读出接收到的数据。

5) 除数锁存器 DLR

DLR 是一个 16 位的寄存器。外部时钟按 DLR 中的除数(分频系数)进行分频, 可以获得所需的波特率。如果外部时钟频率 f 已知, 而 8250 所要求的波特率 B 也已规定, 那么就可以由下式求出 DLR 中除数的值:

$$\text{除数} = f / (B \times 16)$$

通常, 8250 使用 1.843 2 MHz 的基准时钟输入, 所以上式可写为

$$\text{除数} = 1\,843\,200 / (B \times 16)$$

例如, 若要求使用 1 200 波特率来传送数据, 则可计算出除数应为 96。在初始化 8250 时, 最开始就应将除数写到 DLR 中, 以便产生所希望的波特率。为了写入除数, 应首先把 LCR 的 D_7 置 1, 然后将 16 位除数按先低 8 位、后高 8 位的顺序写入 DLR。写完后, 还应把 LCR 的 D_7 再置为 0, 以便 8250 进行正常操作。

6) 中断允许寄存器 IER

IER 只使用 $D_0 \sim D_3$ 这 4 位, 高 4 位不用。 $D_0 \sim D_3$ 每位的 1 或 0 分别用于允许或禁止 8250 的 4 个中断源发出中断请求, 其格式如图 8.43 所示。

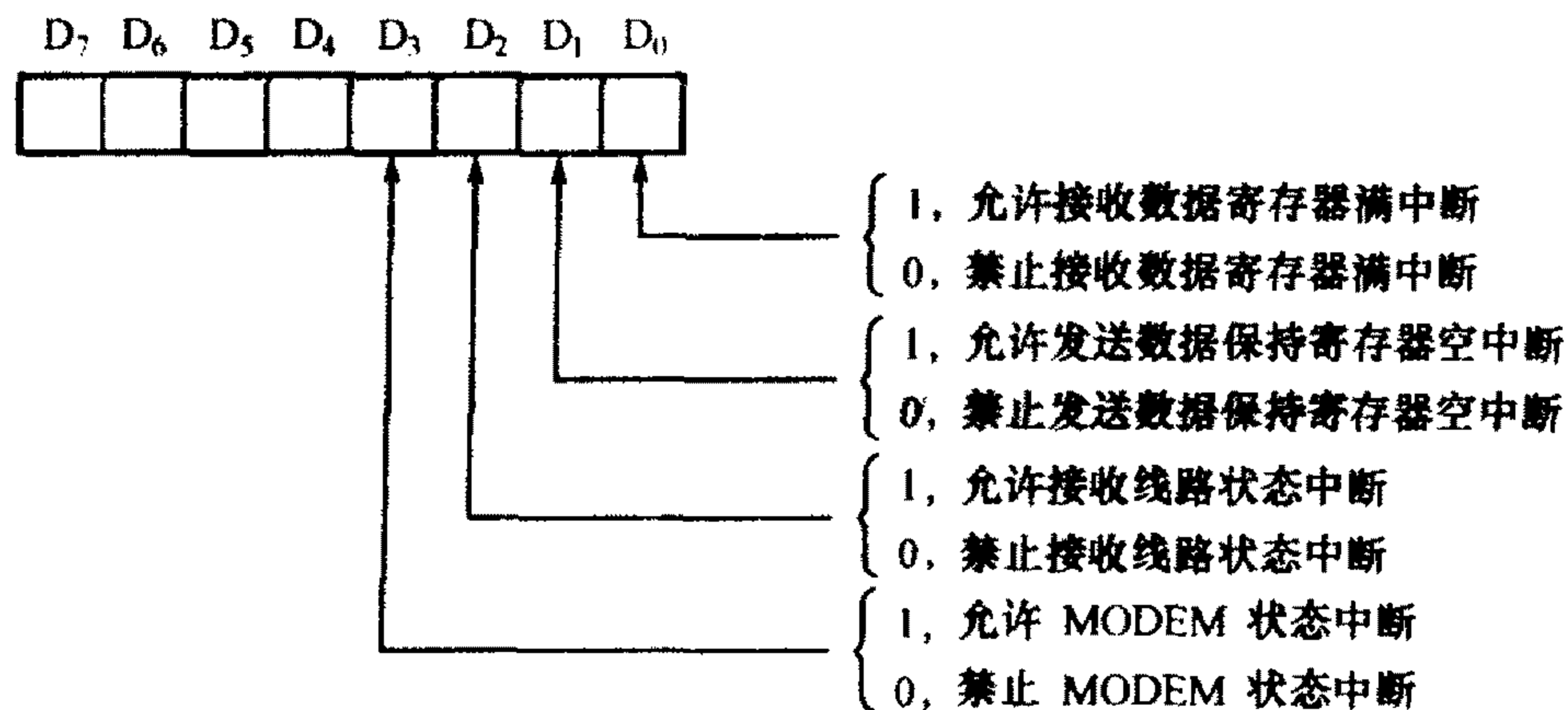


图 8.43 中断允许寄存器 IER

如果 IER 的 $D_0 \sim D_3$ 均为 0, 则禁止 8250 发出中断。在 IER 中, 接收线路状态引起的中断包括越限错、奇偶校验错、结构错和线路间断。

7) 中断识别寄存器 IIR

IIR 为 8 位寄存器, 其高 5 位恒为 0, 只使用低 3 位作 8250 的中断识别标志位。格式如图 8.44 所示。8250 有 4 个中断源, 它们的中断优先级顺序为:

- ① 接收器线路状态中断为最高优先级, 包括越限、奇偶校验错、结构错和线路间断。读 LSR 可使此中断复位。
- ② 第二是接收数据缓冲寄存器满中断。读 RBR 可复位此中断。
- ③ 第三是发送数据保持寄存器空中断。写 THR 可复位此中断。
- ④ 最低优先级为 MODEM 状态中断, 包括 CTS、DSR、RI、DCD 等 MODEM 状态中断源。读 MODEM 状态寄存器可复位此中断。

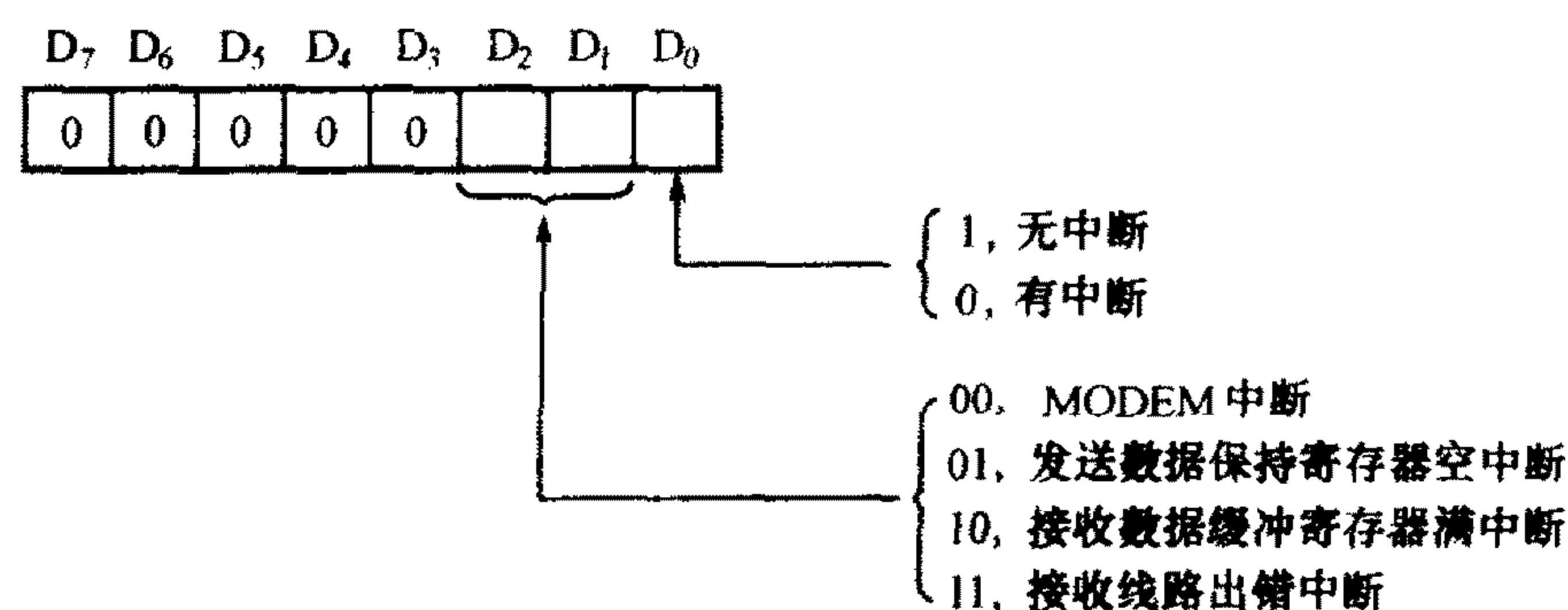


图 8.44 中断识别寄存器 IIR

8) MODEM 控制寄存器 MCR

MCR 是一个 8 位的寄存器, 用来对 MODEM 实施控制。其中高 3 位恒为 0, 其余各位的功能如图 8.45 所示。

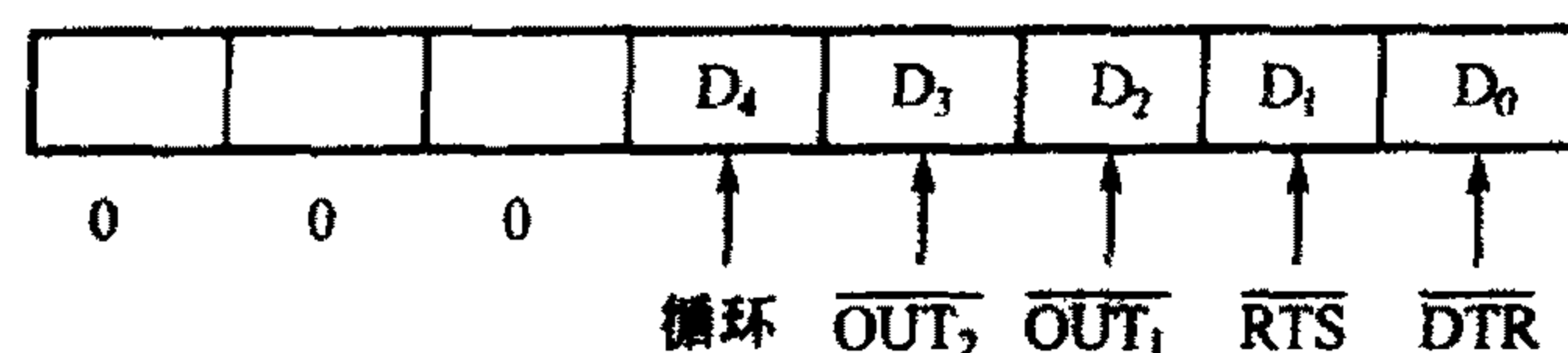


图 8.45 MODEM 控制寄存器 MCR

D_0 用于设置数据终端准备好信号。当它为 1 时, 使 8250 的 \overline{DTR} 输出为低电平, 表示 8250 准备好接收数据; 当它为 0 时, 使 8250 的 \overline{DTR} 输出为高电平, 表示 8250 没有准备好。

D_1 为 1 时, 8250 的 \overline{RTS} 输出低电平, 表示 8250 已准备好发送数据; 当它为 0 时, \overline{RTS} 输

出高电平,表明 8250 未准备好发送。

D_2 、 D_3 分别用于控制 8250 的输出线 $\overline{OUT_1}$ 和 $\overline{OUT_2}$ 。当它们为 1 时,对应的 OUT 输出为 0;而当它们为 0 时,对应的 \overline{OUT} 输出为 1。

D_4 用于环回检测控制,实现 8250 的自我环回测试。当 $D_4 = 1$ 时, SOUT 为高电平状态,而 SIN 将与系统相分离,这时 TSR 的数据将由 8250 内部直接回送到 RSR 的输入端。MODEM 用以控制 8250 的 CTS、DSR、RLSD 和 RI 信号与系统分离。同时,8250 用来控制 MODEM 的输出信号 RTS、DTR、 $\overline{OUT_1}$ 和 $\overline{OUT_2}$ 在 8250 芯片内部与 \overline{CTS} 、 \overline{DSR} 、 \overline{RLSD} 和 \overline{RI} 相连接,实现数据在 8250 芯片内部的自发自收。这样,8250 发送的串行数据在其内部被接收,从而完成 8250 的自检,并且在完成自测试过程中不需要外部连线。在自回环测试时,中断仍能产生。值得注意的是,在这种情况下,MODEM 状态中断是由 MODEM 控制寄存器提供的。

当 $D_4 = 0$ 时,8250 正常工作。从环回测试转到正常工作状态,必须对 8250 重新初始化,其中包括将 D_4 清零。

9) MODEM 状态寄存器 MSR

MSR 用来反映 8250 与通信设备之间应答联络输入信号的当前状态以及这些信号的变化情况。MODEM 状态字的格式如图 8.46 所示。

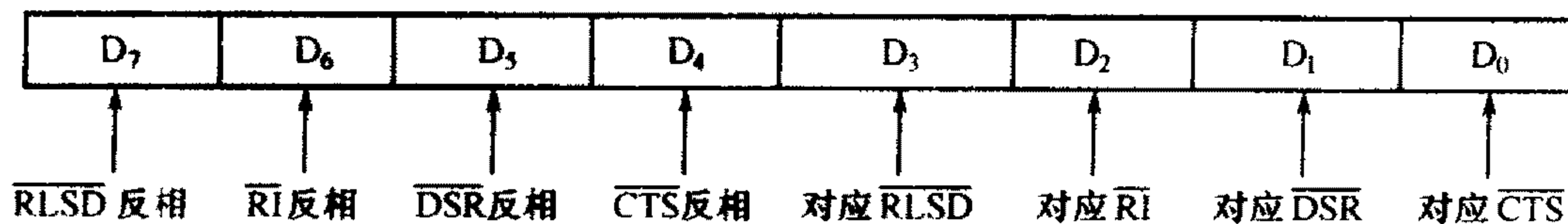


图 8.46 MODEM 状态寄存器 MSR

MSR 的低 4 位是应答输入信号发生变化(从高电平变低电平或从低电平变高电平)的状态标志,CPU 读 MSR 时,把这 4 位同时清零。这 4 位分别对应 \overline{CTS} 、 \overline{DSR} 、 \overline{RI} 和 \overline{RLSD} 。当某位为 1 时,表示从上次读 MSR 后,相应的应答输入信号发生了变化;当某位为 0 时,则说明相应的应答输入信号状态无改变。

MSR 的高 4 位反映了 \overline{CTS} 、 \overline{DSR} 、 \overline{RI} 和 \overline{RLSD} 这 4 个输入信号的当前状态。

- ① D_4 是 \overline{CTS} 反相之后的状态,自测试时为 RTS 的状态;
- ② D_5 是 \overline{DSR} 反相之后的状态,自测试时为 DTR 的状态;
- ③ D_6 是 \overline{RI} 反相之后的状态,自测试时为 $\overline{OUT_1}$ 的状态;
- ④ D_7 是 \overline{RLSD} 反相之后的状态,自测试时为 $\overline{OUT_2}$ 的状态。

3. 8250 的工作过程

下面简单介绍 8250 的工作过程。

1) 数据发送过程

CPU 将要发送的数据以字符为单位写到 8250 的 THR 中。当 TSR 中的数据全部移出变空时,存于 THR 中待发送的数据就会自动并行送到 TSR^①。TSR 在发送时钟的激励下,按照事先和接收方约定的字符传送格式(参见 LCR 寄存器格式,图 8.41),加上起始位、奇偶校验位和停止位,再以约定的波特率(由波特率控制部分产生)按照从低到高的顺序一位接一位地由 SOUT 端发送出去。

一旦 THR 的内容送到 TSR,就会在 LSR 中建立“发送数据保持寄存器空”的状态位;而且也可以用此状态位来触发产生中断。因此,查询该状态位或者利用该状态触发的中断,即可实现数据的连续发送。

2) 数据接收

由通信对方来的数据在接收时钟 RCLK 作用下,通过 SIN 端逐位进入 RSR。RSR 根据初始化时定义的数据位数确定接收到了一个完整的数据后会立即将数据自动并行传送到 RBR。RBR 收到 RSR 的数据后,就立即在状态寄存器中建立“接收数据准备好”的状态,而且也可以用此状态位来触发中断。因此,查询该状态位或者利用该状态触发的中断,即可实现数据的连续接收。

由于串行异步通信的速率较低,无论是用查询方式或中断方式来实现异步通信均不很困难。

4. 8250 的应用

1) 8250 的寻址和连接

一片 8250 芯片共占用 7 个接口地址。表 8-8 详细列出了各内部寄存器具体的地址安排,另外还列出了 IBM PC/XT 中异步串行通信口 COM1 各寄存器的物理地址(COM2 的物理地址相应为 2F8H~2FFH)。

8250 内部有 10 个与编程使用有关的寄存器,可利用选片信号 CS_0 、 CS_1 和 $\overline{CS_2}$ 选中 8250,利用芯片上 A_0 、 A_1 、 A_2 3 根地址线的八种不同编码选择 8 个寄存器,再利用通信控制字的最高位——除数锁定位(DLAB)来选中除数锁存器。由于有的寄存器是只写的,有的寄存器是只读的,故还可以利用读/写信号来加以选择。通过上述这些办法,就可以对指定的寄存器进行寻址访问。

在 PC 机中,串行通信接口由 8250 来实现,图 8.47 表示了它与总线的连接。由图可知,8250 的地址由 10 条地址线来决定,其地址范围为 3F8H~3FFH(COM1)。在寻址 8250 时, \overline{AEN} 信号总处于低电平。由于 \overline{ADS} 始终接地, CS_0 和 CS_1 接高电平,故只要地址译码输出使

^① 8250 初始化后,TSR 为空状态,所以初始化后传送到 THR 的第一个字符总是立即送入 TSR。

\overline{CS}_2 为低电平即可选中 8250。再利用表 8-8 所示的寻址方法,就可对 8250 的 9 个内部寄存器寻址。表中 DLAB 为 LCR 的最高位。

表 8-8 8250 内部寄存器寻址

CS_0	CS_1	\overline{CS}_2	A_2	A_1	A_0	DLAB	COM1 地址	寄存器
1	1	0	0	0	0	0	3F8H	发送数据保持寄存器 THR(写),接收数据缓冲寄存器 RBR(读)
1	1	0	0	0	0	1	3F8H	除数锁存器(低 8 位)DLL
1	1	0	0	0	1	1	3F9H	除数锁存器(高 8 位)DLH
1	1	0	0	0	1	0	3F9H	中断允许寄存器 IER
1	1	0	0	1	0	×	3FAH	中断识别寄存器 IIR
1	1	0	0	1	1	×	3FBH	通信线路控制寄存器 LCR
1	1	0	1	0	0	×	3FCH	MODEM 控制寄存器 MCR
1	1	0	1	0	1	×	3FDH	通信线路状态寄存器 LSR
1	1	0	1	1	0	×	3FEH	MODEM 状态寄存器 MSR
1	1	0	1	1	1	×	3FFH	无效

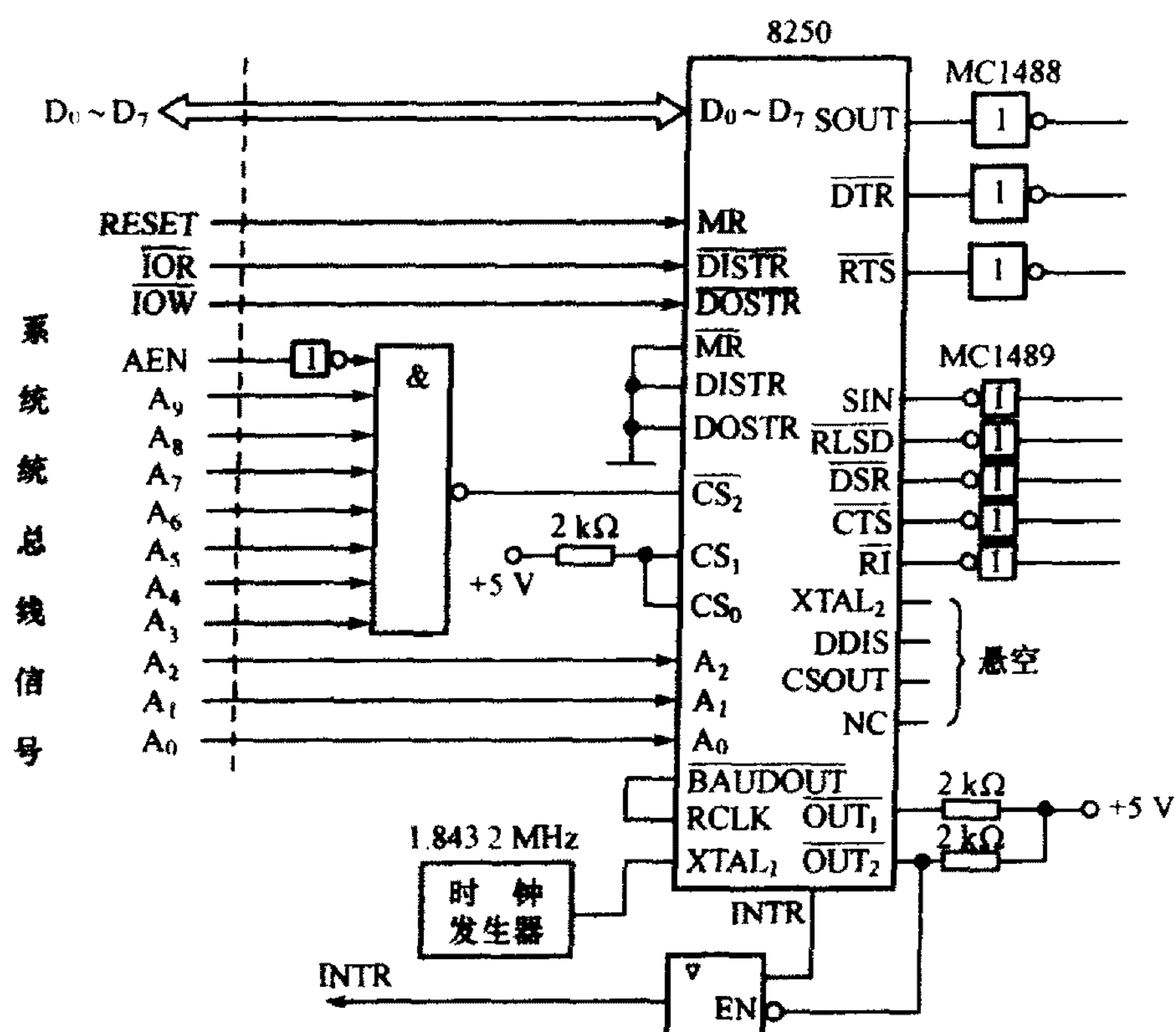


图 8.47 8250 与系统的连接

时钟发生器将外部时钟信号由 XTAL₁ 加到 8250 上,而其 BAUDOUT 输出又作为接收时钟加到 RCLK 上。芯片上的一些引脚固定接高电平或接地,而一些不用的则悬空。这是 8250 在电路连接上具有的灵活性。

2) 初始化及其应用

8250 初始化时,通常首先使通信控制字的 D₇ = 1,即使 DLAB 为 1。在此条件下,将除数低 8 位和高 8 位分别写入 8250 内部的除数锁存器。然后再以不同的地址分别写入通信控制字、MODEM 控制字及中断允许字等。其具体做法可按图 8.48 所示的流程依次进行。现以图 8.47 所示电路为例,对 8250 进行初始化编程。

假定所需的波特率为 1 200 波特,数据格式为:1 位停止位,7 位数据位,奇校验。

初始化程序如下:

```

START: MOV DX,3FBH      ;LCR 的地址
        MOV AL,80H      ;开始
        OUT DX,AL       ;使 LCR 的 D7 = 1
        MOV DX,3F8H     ;DLL 的地址
        MOV AL,60H      ;除数为 0060H
        OUT DX,AL       ;写除数低 8 位
        INC DX          ;DLH 的地址
        MOV AL,0        ;
        OUT DX,AL       ;写除数高 8 位
        MOV DX,3FBH     ;LCR 的地址
        MOV AL,0AH      ;1 位停止位,7 位数据位,奇校验
        OUT DX,AL       ;初始化通信控制寄存器
        MOV DX,3FCH     ;MCR 的地址
        MOV AL,03H      ;使 DTR 和 RTS 有效
        OUT DX,AL       ;初始化 MODEM 控制器
        MOV DX,3F9H     ;IER 的地址
        MOV AL,0        ;禁止所有中断
        OUT DX,AL       ;写中断允许寄存器
        .....

```

上面的初始化程序是完全按照图 8.48 所示的顺序编写的,即首先写除数锁存器,而要

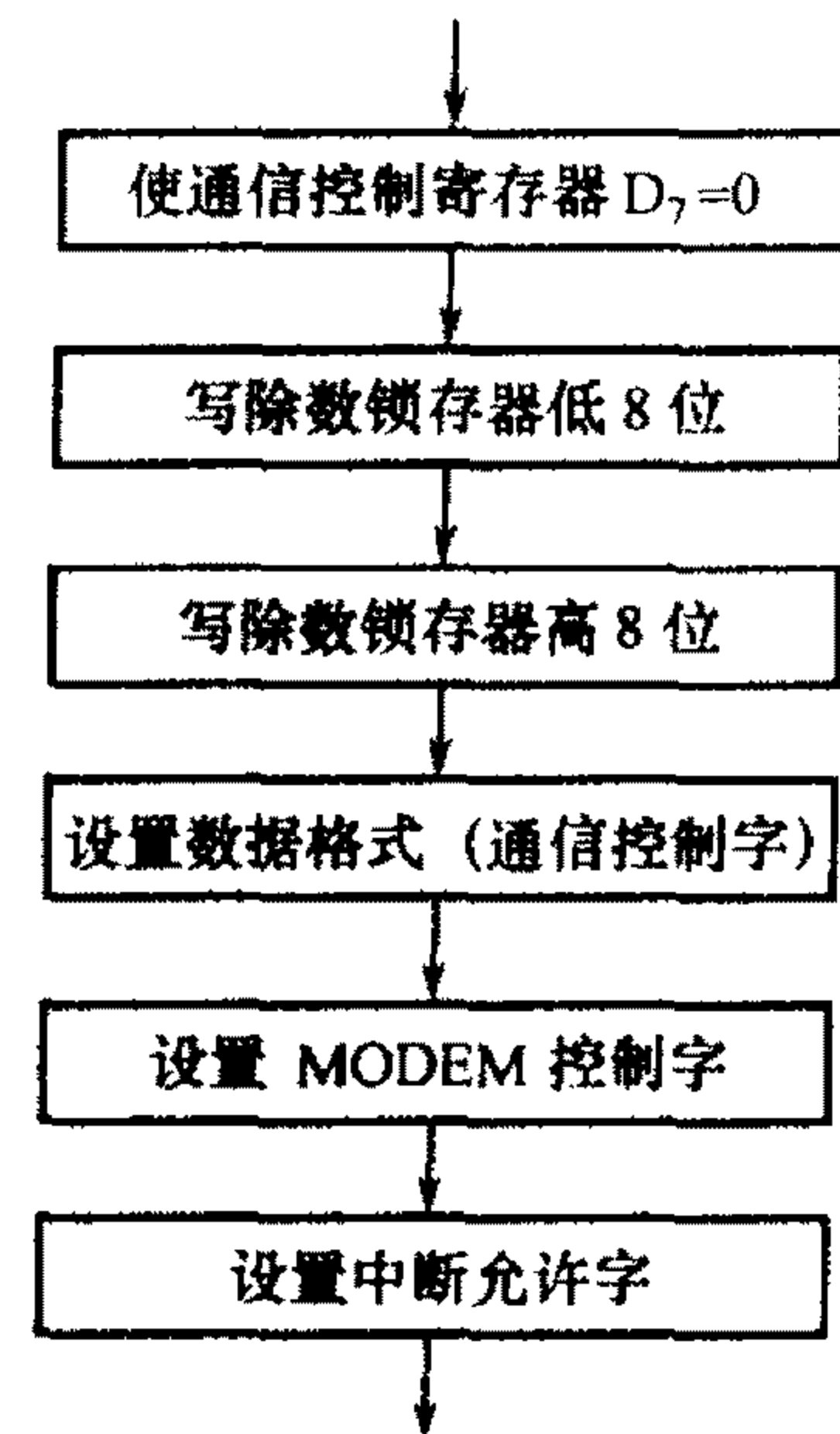


图 8.48 8250 的初始化流程图

将除数写入,先要使通信控制寄存器的 $D_7 = 1$,亦即 $DLAB = 1$,然后再写入 16 位的除数 0060H,即十进制数 96。由于加在 $XTAL_1$ 上的时钟频率为 1.843 2 MHz,故波特率为 1 200 波特。

初始化通信控制字为 00001010B。它指定数据为 7 位,停止位为 1 位,奇校验。MODEM 控制字为 00000011B,使 \overline{DTR} 和 \overline{RTS} 均为低电平,即有效状态。最后,将中断允许控制字写入中断允许寄存器。由于中断允许字为 00H,故禁止 4 个中断源可能形成的中断。8250 的中断,在硬件上是通过 \overline{OUT}_2 输出控制的三态门接到 8259 上去的。若允许中断,则一方面要使 \overline{OUT}_2 输出为低电平,同时,再初始化中断允许寄存器。 \overline{OUT}_2 是由 MODEM 控制字的 D_3 来控制。只有当 MODEM 控制字的 $D_3 = 1$ 时, \overline{OUT}_2 才为低电平。上述的 MODEM 控制字为 03H,其 $D_3 = 0$,故 $\overline{OUT}_2 = 1$,这时禁止中断请求输出。

发送数据的程序接在初始化程序之后。若采用查询方式发送数据,且假定要发送的字节数放在 BX 中,要发送的数据顺序存放在以 DATA 为首地址的内存区域中,则发送数据的程序段如下:

```
SENDPRG:      MOV DX,3FDH
               LEA SI,DATA
WAITTHR:      IN AL,DX
               TEST AL,20H           ;检查 THR 是否空
               JZ WAIT_SE
               PUSH DX
               MOV DX,3F8H
               LODSB
               OUT DX,AL             ;发送一个字节
               POP DX
               DEC BX
               JNZ WAIT_THR
               ....
```

同样,在初始化后,可以利用查询方式实现数据的接收。下面是接收一个数据的程序段:

```
RECVPRG:      MOV DX,3FDH
WAITRBR:      IN AL,DX
               TEST AL,1EH           ;检查是否有任何错误产生
               JNZ ERROR
               TEST AL,01H           ;检查数据准备好否
               JZ WAITRBR
```

```
MOV DX,3F8H
IN AL,DX           ;接收一个字节
AND AL,7FH        ;只保留低 7 位
.....
ERROR:            .....
```

该程序首先测试状态寄存器,看接收的数据是否有错。若有错,就转向错误处理 ERROR;若无错时,再看是否已收到一个完整的数据,是,则从 8250 的接收数据缓冲寄存器中读出,并取事先约定的 7 位数据,将其放在 AL 中。

除查询方式外,也可以利用中断方式实现数据的串行异步传送。若假设系统以查询方式发送数据,以中断方式接收数据,则控制程序包括三部分:对 8250 的初始化、在初始化完时(假如其他接口初始化在此之前)开中断、接收字符的中断服务程序(接收到一个字符时自动调用此程序)。

当 8250 的接收数据缓冲寄存器满而产生中断时,中断请求经过中断控制器 8259 送给 CPU。CPU 中断响应后,转向中断服务程序。该中断服务程序首先进行断点和现场保护。再取回接收状态和接收到的一个字符,并检查接收有无差错。若有错,则进行错误处理;无错,则将接收到的字符保存在指定的接收缓冲区中。再使缓冲区地址指针增量,准备存放下一个接收到的字符。然后恢复现场和断点,开中断并中断返回。这里特别说明的是,在中断服务程序结束前,必须向 8259 发送一个中断结束命令 EOI,使 8259 将其中断服务寄存器的状态复位,以便又能处理其他低级别的中断。另外,还要考虑接收缓冲区的溢出问题。可将缓冲区设置为一个环形结构,即指针每次加 1 后判断是否超出缓冲区最大地址值,若超出,则将指针调整为缓冲区起始地址值。该结构的接收缓冲区解决了指针越界问题,但要求对接收到的数据的处理速度要高于接收速度,否则前面的数据还未来得及处理就会被后面接收的数据覆盖。

因篇幅所限,有关这方面详细的程序设计就请读者自行设计或查阅其他相关书籍。

8.3 模拟量输入/输出接口

模拟量输入/输出通道是微型计算机与控制对象之间的一个重要接口,也是实现语音、视频处理和工业过程控制的重要组成部分。

在工业生产中,需要测量和控制的物理量往往是连续变化的量,如电流、电压、温度、压力、位移、流量等。为了利用计算机实现对工业生产过程的自动监测和控制,首先必须要能够将生产过程中监测设备输出的连续变化的模拟量转变为计算机能够识别和接受的数字

量。其次,还要能够将计算机发出的控制命令转换为相应的模拟信号,去驱动模拟调节执行机构。这样两个过程,就需要模拟量的输入和输出通道来完成。

8.3.1 模拟量输入/输出通道

模拟量输入/输出通道的结构如图 8.49 所示。下面分别介绍输入和输出通道中各环节的作用。

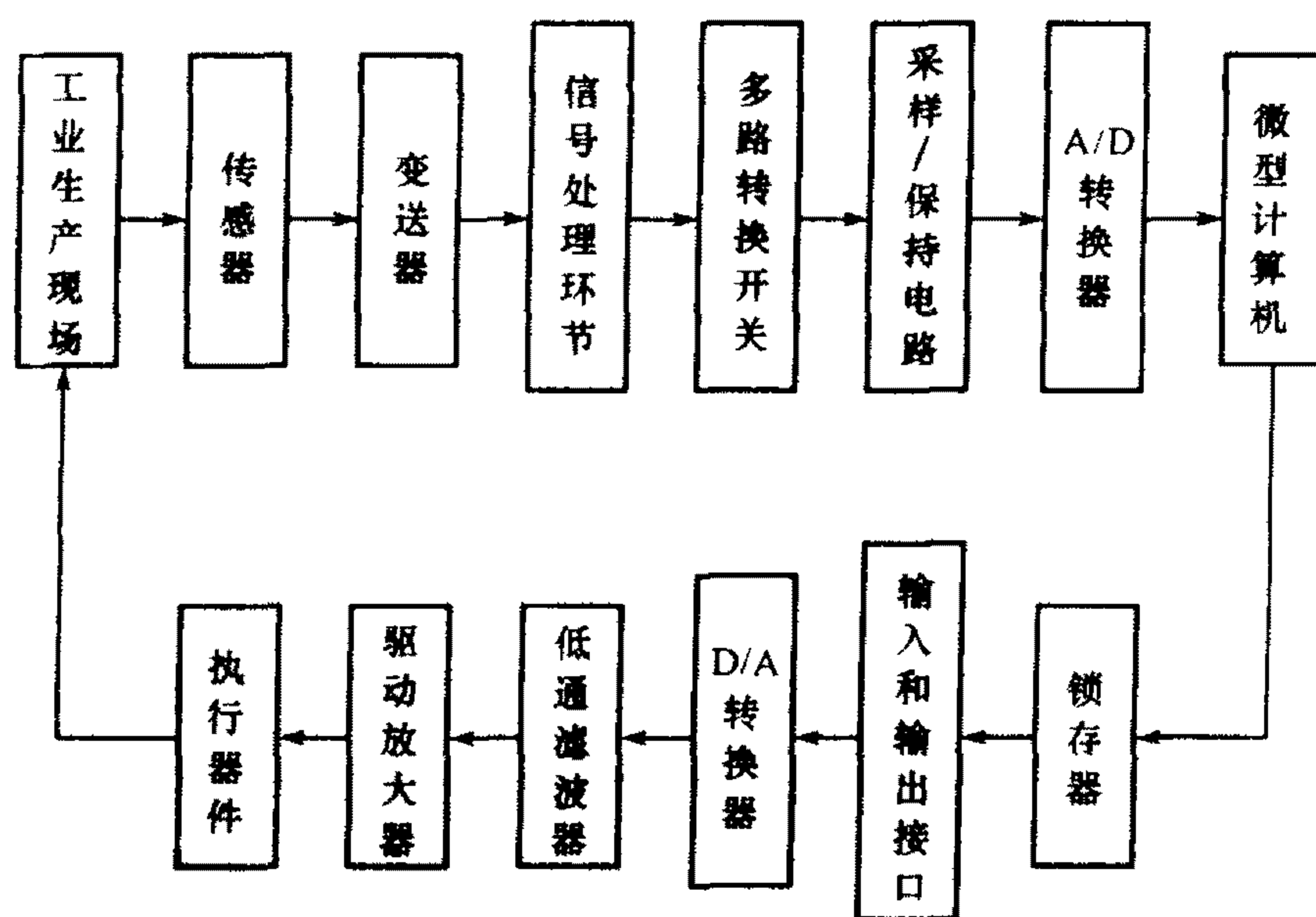


图 8.49 模拟量的输入、输出通道结构图

1. 模拟量的输入通道

典型的模拟量输入通道由以下几部分组成：

1) 传感器(Transducer)

传感器是用于将工业生产现场的某些非电物理量转换为电量(电流、电压)的器件。例如,热电偶能够将温度这个物理量转换成几毫伏或几十毫伏的电压信号,所以可用它作为温度传感器;而压力传感器可以把物理量压力的变化转换为电信号,等等。

2) 变送器

一般来讲,传感器输出的电信号都比较微弱,有些传感器的输出甚至是电阻值、电容值等。为了易于与信号处理环节衔接,就需要将这些微弱电信号及电阻值等转换成一种统一的电信号,变送器就是实现这一功能的器件。它将传感器的输出信号转换成 $0 \sim 10 \text{ mA}$ 、 $4 \sim 20 \text{ mA}$ 的统一电流信号或者 $0 \sim 5 \text{ V}$ 的电压信号。

3) 信号处理环节

信号处理环节主要包括信号的放大及干扰信号的去除。它将变送器输出的信号进行放大或处理成与 A/D 转换器所要求的输入相适应的电压。另外,传感器通常都安装在现场,环境比较恶劣,其输出常叠加有高频干扰信号。因此,信号处理环节通常是低通滤波电路,如 RC 滤波器或由运算放大器构成的有源滤波电路等。

4) 多路转换开关(Multiplexer)

在生产过程中,要监测或控制的模拟量往往不止一个,尤其是数据采集系统中,需要采集的模拟量一般比较多,而且不少模拟量是缓慢变化的信号。对这类模拟信号的采集,可采用多路模拟开关,使多个模拟信号共用一个 A/D 转换器进行采样和转换,以降低成本。

5) 采样/保持电路(Sample Holder)

在数据采样期间,保持输入信号不变的电路称为采样/保持电路。由于输入模拟信号是连续变化的,而 A/D 转换器完成一次转换需要一定的时间,这段时间称为转换时间。不同的 A/D 转换芯片,其转换时间不同。对变化较快的模拟输入信号,如果不在转换期间保持输入信号不变,就可能引起转换误差。A/D 转换芯片的转换时间越长,对同样频率模拟信号的转换精度的影响就越大。所以,在 A/D 转换器前面要增一级采样/保持电路,以保证在转换过程中输入信号保持在其采样时的值不变。

6) A/D 转换器(Analog - Digital Converter)

这是模拟量输入通道的中心环节,它的作用是将输入的模拟信号转换成计算机能够识别的数字信号,以便计算机进行分析和处理。

2. 模拟量的输出通道

计算机的输出信号是数字信号,而有的控制执行元件要求提供模拟的输入电流或电压信号,这就需要将计算机输出的数字量转换为模拟量;这个过程的实现由模拟量的输出通道来完成。输出通道的核心部件是数模(Digital - Analog Converter, D/A)转换器。由于将数字量转换为模拟量同样需要一定的转换时间,也就要求在整个转换过程中待转换的数字量要保持不变;而计算机的运行速度很快,其输出的数据在数据总线上稳定的时间很短,因此,在计算机与 D/A 转换器之间必须加一级锁存器,以保持数字量的稳定。D/A 转换器的输出端一般还要加上低通滤波器,以平滑输出波形。另外,为了能够驱动执行器件,还需要将输出的小功率的模拟量加以放大。

8.3.2 数模(D/A)转换器

1. D/A 转换器的基本原理

D/A 转换器的作用是将数字量转换为相应的模拟量。数字量由二进制位组成,每个二进制位的权为 2^i ,要把数字量转换为相应的模拟量电压(多数情况需要转换后的模拟信号以

电压的形式输出), 需要先把数字量的每一位上的代码按权转换成对应的模拟电流, 再把模拟电流相加, 最后由运算放大器将其转变成模拟电压。将数字量转换成对应模拟电流的工作由 D/A 转换器来完成。

典型的 D/A 转换器芯片通常由模拟开关、电阻网络以及缓冲电路等组成。其框图如图 8.50 所示。

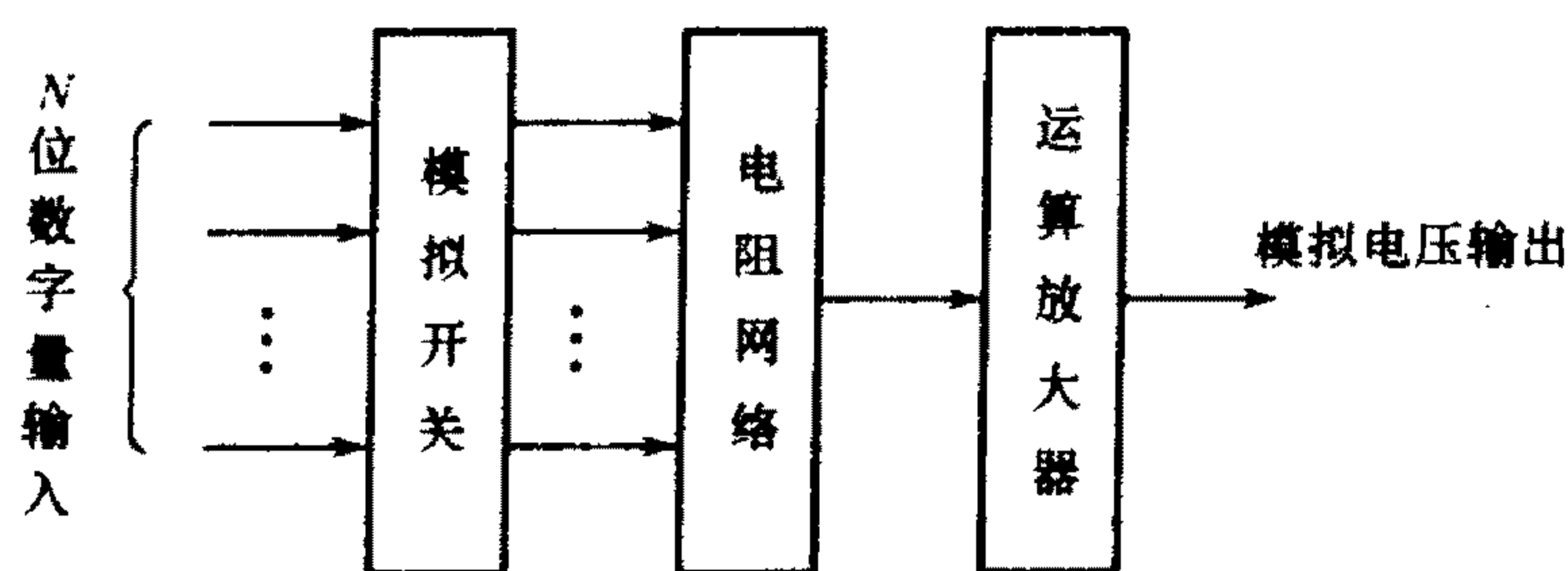


图 8.50 D/A 转换器结构示意图

电阻网络是 D/A 转换器的核心部件。其结构有权电阻网络和 $R-2R$ T 形电阻网络两种主要形式。

下面先复习运算放大器的原理, 然后从中引申出 D/A 转换器的工作原理。

运算放大器具有如下特点:

- ① 开环放大倍数很高(一般为几千到几十万倍), 因此所需输入电压很小;
- ② 输入阻抗非常大, 所以其输入电流很小;
- ③ 输出阻抗非常小, 使运算放大器的负载能力很强。

一个简单的运算放大器电路如图 8.51 所示。

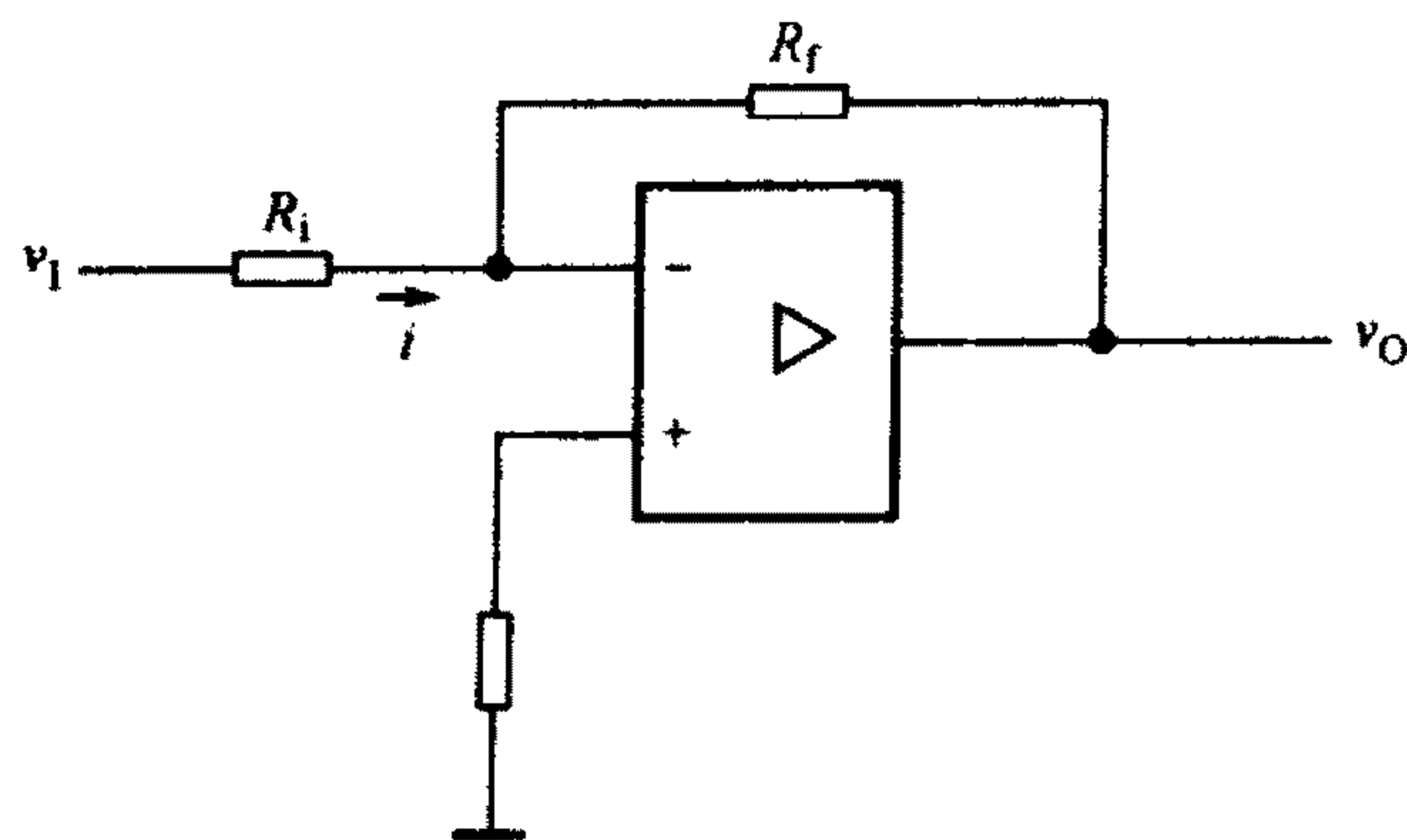


图 8.51 基本运算放大器电路

对运算放大器来说, 其输出电压 v_o 与输入电压 v_i 之间有如下关系:

$$V_O = - \frac{R_f}{R_i} V_I \quad (9.1)$$

式中, R_f 为运算放大器的反馈电阻, R_i 为输入电阻。

若输入端有 n 个支路, 如图 8.52 所示, 则输入与输出的关系可表示为

$$V_O = - R_f \sum_{j=1}^n \frac{1}{R_j} V_I \quad (9.2)$$

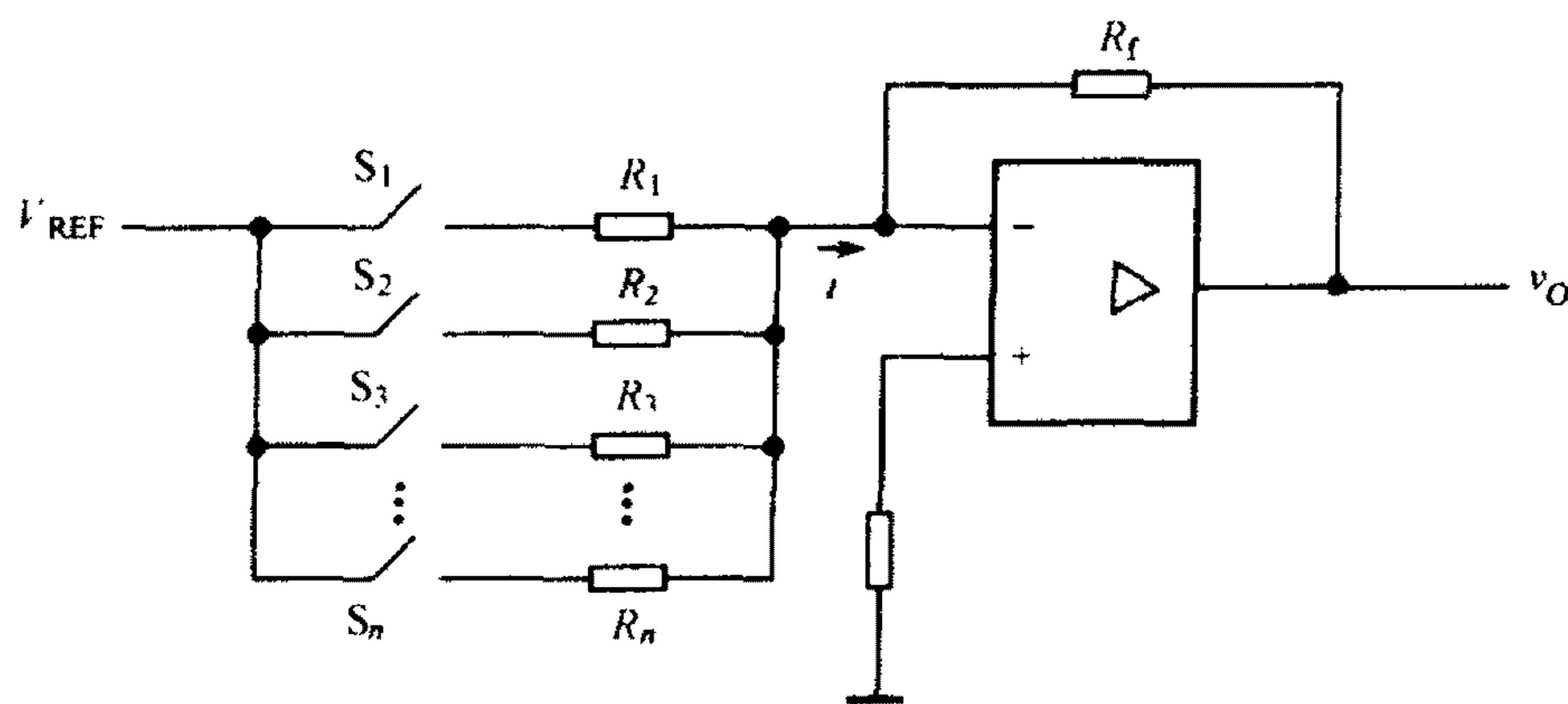


图 8.52 多路输入的运算放大器电路

如果使各支路上的输入电阻 R_1, R_2, \dots, R_n 分别等于 $2^1 R, 2^2 R, \dots, 2^n R$, 即每一位电阻值都具有权值 2^j (j 为该电阻所在的位数), 且由一个开关 S_j 来控制, 当 S_j 合上时, $S_j = 1$; S_j 断开时 $S_j = 0$, 并令 $V_{REF} = \frac{R_f}{R} V_I$, 则可得出输出电压 v_O 和输入的关系为:

$$v_O = - \sum_{j=1}^n \frac{1}{2^j} S_j V_{REF} \quad (9.3)$$

通过式(9.3)可以看出:

① 当所有开关 S_j 断开时, $v_O = 0$;

② 当所有开关 S_j 闭合时, 输出电压 v_O 为最大, 即 $v_O = \frac{2^j - 1}{2^j} V_{REF}$ 。

如果用二进制编码来控制图 8.52 中每一路的 S_j , 当第 i 路的二进制码为 1 时, 使第 j 位的 S_j 闭合; 第 j 路的二进制码为 0 时, 使对应的 S_j 断开, 则数字量的变化就转换成了模拟量的变化。这就是 D/A 转换的基本原理。

D/A 转换器的转换精度与基准电压 V_{REF} 和权电阻的精度以及数字量的位数 j 有关。显然, 位数越多, 转换精度就越高, 但同时所需的权电阻的种类就越多。由于在集成电路中制造高阻值的精密电阻比较困难, 因此常用 $R-2R$ T 形电阻网络来代替权电阻网络, 如图 8.53 所示。这是一个简化了的 T 形电阻网络原理图, 它只由两种阻值 R 和 $2R$ 组成, 用集成

工艺生产较为容易,精度也容易保证,因此得到比较广泛的应用。

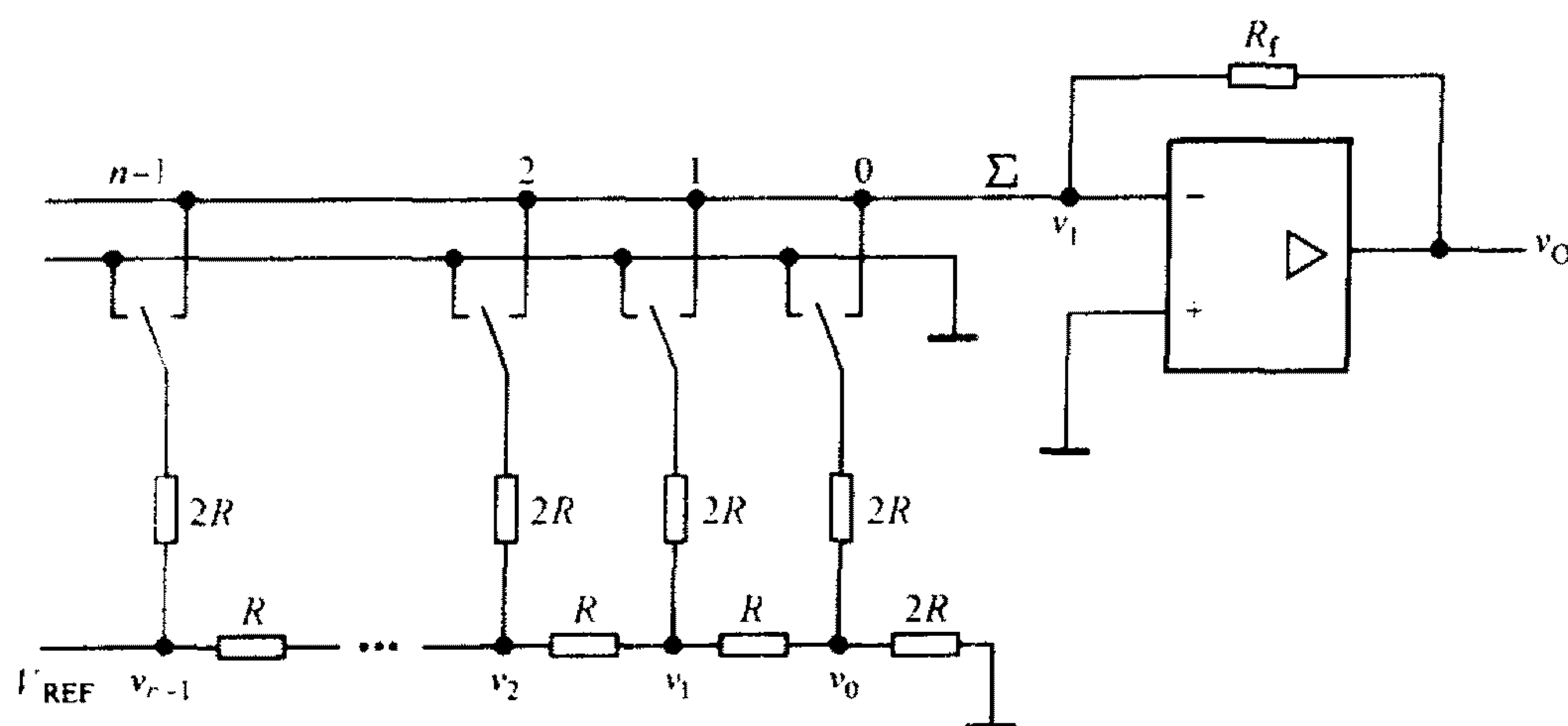


图 8.53 R-2R T形电阻网络

R-2R T形电阻网络的输出和输入电压的关系表达式为

$$V_O = -\frac{D}{2^j} \times \frac{R_f}{R} \times V_{REF} \quad (9.4)$$

式中, D 为输入的数字量, j 为数字量的位数。

由上式可知,输出电压 V_O 正比于输入数字量 D ,而幅度大小由 V_{REF} 和 R_f/R 的比值决定。若使 $R_f/R=1$,并且输入为 8 位的数字量,则上式简化为

$$V_O = -\frac{D}{256} \times V_{REF} \quad (9.5)$$

这就是 8 位 D/A 转换器的输出电压与数字量的关系式。

电阻网络是构成 D/A 转换器的主要部件,但在具体电路中还需要一些其他部件。一个实际的 D/A 转换器原理框图如图 8.54 所示。

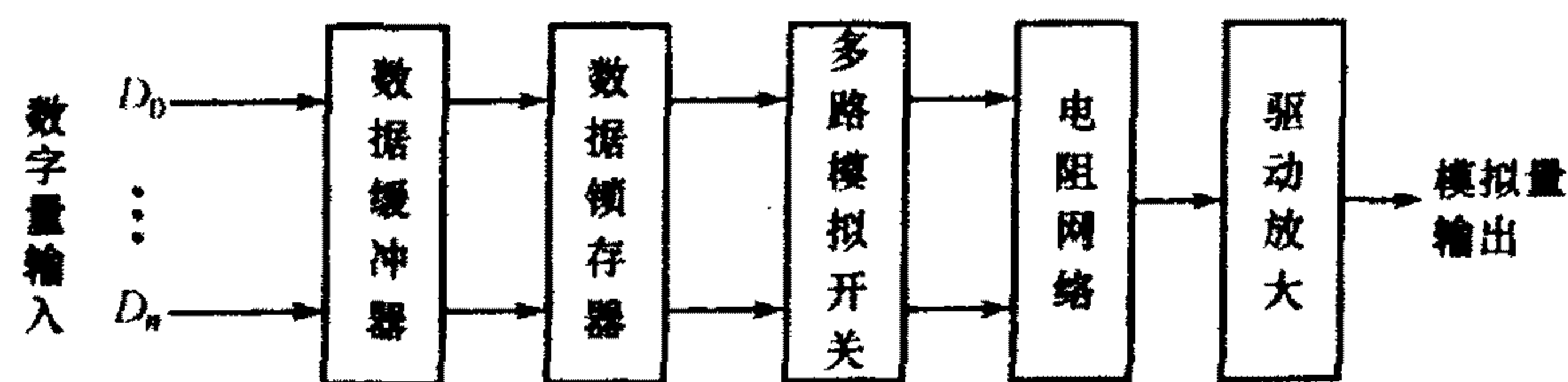


图 8.54 D/A 转换器原理框图

首先将待转换的数字量 $D_0 \sim D_n$ 通过数据缓冲器送至数据锁存器,以确保在整个转换过程中数字量的稳定(仅在一次转换过程结束后,才允许将新的数字量存入)。锁存器的输

出接到多路模拟开关,使数据信号的高、低电平转变成相应的开关状态。各位模拟开关输出的电流通过电阻网络进行加权,合成一个与输入数字量等效的模拟电流信号,再经过驱动放大电路,形成模拟量的输出。

有时,需要 D/A 转换器输出电压信号,对这种情况,可在其输出端接一个运算放大器,将电流信号转换为电压信号输出。

D/A 转换器的输出形式有电压、电流两大类。电压输出型的 D/A 转换器的输出电压一般为 0~5V 或 0~10V,它相当于一个电压源,内阻较小,可带动较大的负载;而电流输出型的则相当于一个电流源,内阻较大,与之匹配的负载电阻不能太大。

2. D/A 转换器的主要技术指标

1) 分辨率(Resolution)

分辨率是 D/A 转换器对数字输入量变化的敏感程度的度量。它表示输入每变化一个最低有效位使输出变化的程度,可用数字量的位数来表示,如 8 位、10 位等;也可定义为输入数字量等于 1 时的电压值与输入数字量等于最大值时的满度电压值之比,例如,对一个 n 位的 D/A 转换器,若其满度电压值为 V ,其最低有效位对应的电压值就为 $V/(2^n - 1)$,则该 D/A 转换器的分辨率等于 $1/(2^n - 1)$,如果用百分比表示,则为 $[1/(2^n - 1)] \times 100\%$ 。

2) 转换精度

转换精度表示由于 D/A 转换器的引入而使其输出和输入之间产生的误差。可用绝对转换精度或相对转换精度来表示。

绝对转换精度是指实际的输出值与理论值之间的差距。它与 D/A 转换器参考电压的精度、权电阻的精度等有关。

相对转换精度是绝对转换精度与满量程输出之百分比,是常用的描述输出电压接近理想值程度的物理量,更具有实用性。例如,一个 D/A 转换器的绝对转换精度为 0.05 V,若输出满刻度值为 5 V,则其相对转换精度为 1%。

与 D/A 转换器转换精度有关的指标还有以下几点:

- ① 温度系数误差 在允许范围内,温度每变化 1℃所引起的输出变化。
- ② 电源波动误差 由于电源的波动引起的输出变化。
- ③ 运算放大器误差 与 D/A 转换器相连的运算放大器带来的误差。
- ④ 参考电源误差 由于参考电源的波动引起的输出变化。

需要注意的是,由于不可能用有限位数的数字量来表示连续的模拟量,所以由位数产生的转换误差是不能消除的,是系统固有的。为了尽量减小分辨率造成的转换误差,在系统设计时,应这样来选择 D/A 转换器的位数,使其最低有效位的变化所引起的误差远远小于 D/A 转换器芯片的总误差。

3) 转换时间

转换时间是指当输入数字量满刻度变化(如全 0 到全 1)时,从数字量输入到输出模拟量达到与终值相差 $(1/2)$ LSB(最低有效位)相当的模拟量值所需的时间。它表征了一个 D/A 转换器芯片的转换速率。

4) 线性误差

在 D/A 转换时,若数据连续转换,则输出的模拟量应该是线性的。即在理想情况下,D/A 转换器的输入/输出曲线是一条直线,但实际的输出特性曲线与理想的曲线之间存在一定的误差。把实际输出特性偏离理想转换特性的最大值称为线性误差。通常用这个最大差值折合成的数字量来表示。

例如,一个 D/A 转换器的线性误差小于 $(1/2)$ LSB,表示用它进行 D/A 转换时,其输出模拟量与理想值之差最大不会超过 $(1/2)$ LSB 的输入量产生的输出值。

5) 动态范围

D/A 转换器的动态范围是指最大和最小输出值范围。一般决定于参考电压 V_{REF} 的高低。参考电压高,动态范围就大。整个 D/A 转换电路的动态范围除与 V_{REF} 有关外,还与输出电路的运算放大器的级数及连接方法有关。适当地选择输出电路,可在一定程度上增加转换电路的动态范围。

3. 典型的 D/A 转换器芯片及其应用

D/A 转换器的种类繁多,在目前常用的 D/A 转换芯片中,从数码位数上看,有 8 位、10 位、16 位等;在输出形式上,有电流输出和电压输出;从内部结构上,又可分为含数据输入寄存器和不含数据输入寄存器两类。对内部不含数据输入寄存器的芯片,亦即不具备数据的锁存能力,是不能直接与系统总线连接的。因为对 D/A 转换器来讲,当有数字量输入时,其输出端随之有模拟电流或电压信号建立;而当输入端数字量消失时,输出模拟量也随之消失。另外,为实现对某个对象的控制,要求输出模拟量要能够保持一段时间。在微型计算机系统中,D/A 转换器的输入数据来自 CPU,8088CPU 在执行输出指令时,数据在数据总线上只能维持两个时钟周期,这使得转换后的模拟量在输出端保持时间太短,无法满足实际控制系统的要求。所以,在这类芯片(如 AD7520、AD7521 等)与 CPU 连接时,要在其与 CPU 之间增加数据锁存器(如 74LS273)。而内部已包含数据输入寄存器的 D/A 转换器芯片可直接与系统总线相连,常见的有 DAC0832、AD7524 等。

尽管 D/A 转换器的型号很多,但它们的基本工作原理和功能都是一致的。下面以较常用的 DAC0832 为例,来说明数模转换器与 CPU 的连接方法及其应用。

1) 引出线及内部结构

DAC0832 是一个 8 位的数模转换芯片,内部包含一个 T 形电阻网络,输出为差动电流信号。因此,要想得到模拟电压输出,必须外接运算放大器。其外部引脚图和内部结构图分别如图 8.55 和图 8.56 所示。

DAC0832 各引出线的定义如下:

$D_0 \sim D_7$ 8 位数据输入端;

\overline{CS} 片选信号,低电平有效;

ILE 输入寄存器选通命令,它与 \overline{CS} 、 \overline{WR}_1 一起将要转换的数据送入输入寄存器;

\overline{WR}_1 输入寄存器的写入控制信号,低电平有效;

\overline{WR}_2 数据转换(DAC)寄存器写入控制信号,低电平有效;

\overline{XFER} 传送控制信号,低电平有效,它与 \overline{WR}_2 一起把输入寄存器的数据装入到数据转换寄存器;

I_{OUT1} 模拟电流输出端,当 DAC 寄存器中内容为 0FFH 时, I_{OUT1} 电流最大;当 DAC 寄存器中内容为 00H 时, I_{OUT1} 电流最小;

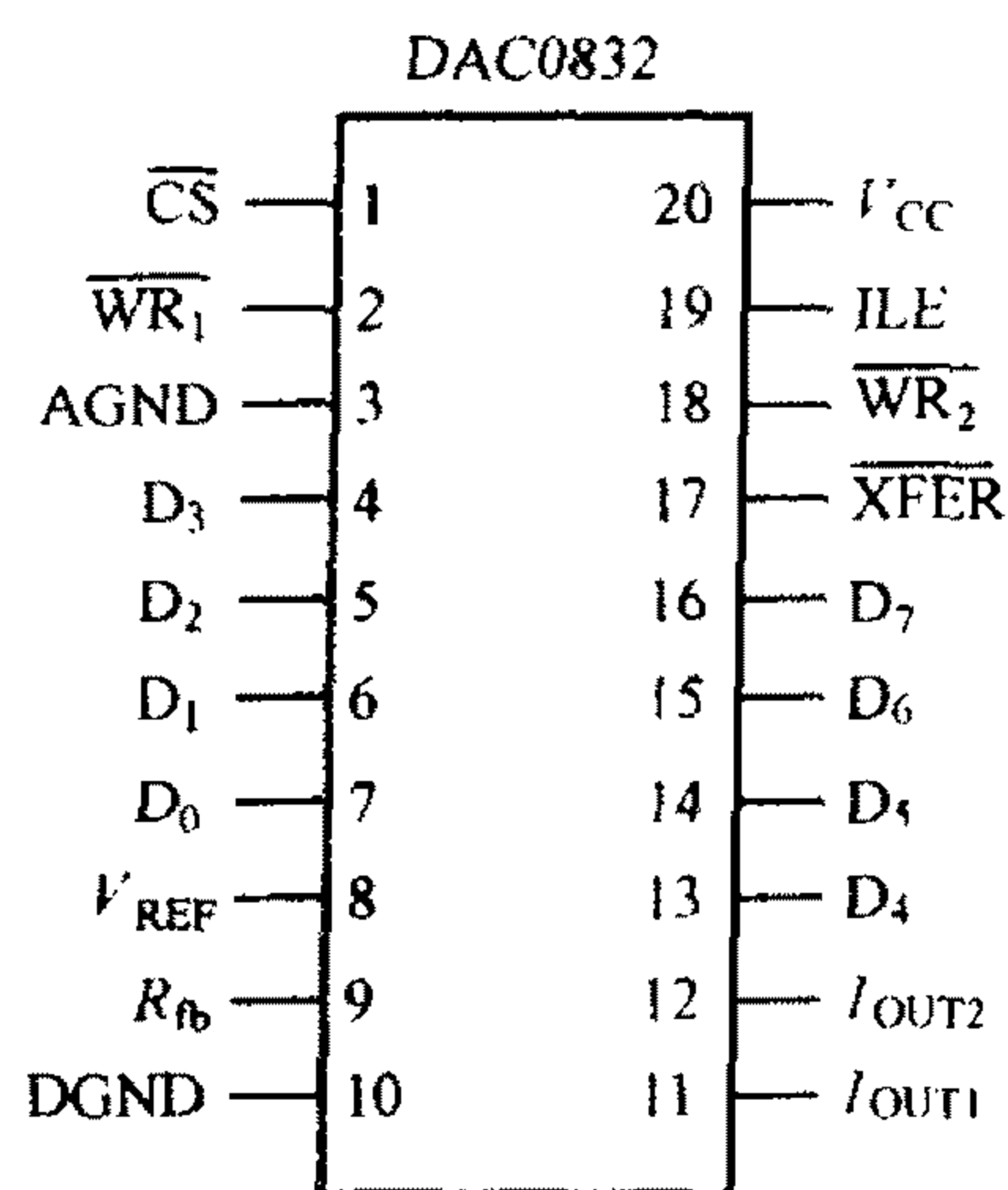


图 8.55 DAC0832 的外部引出线排列图

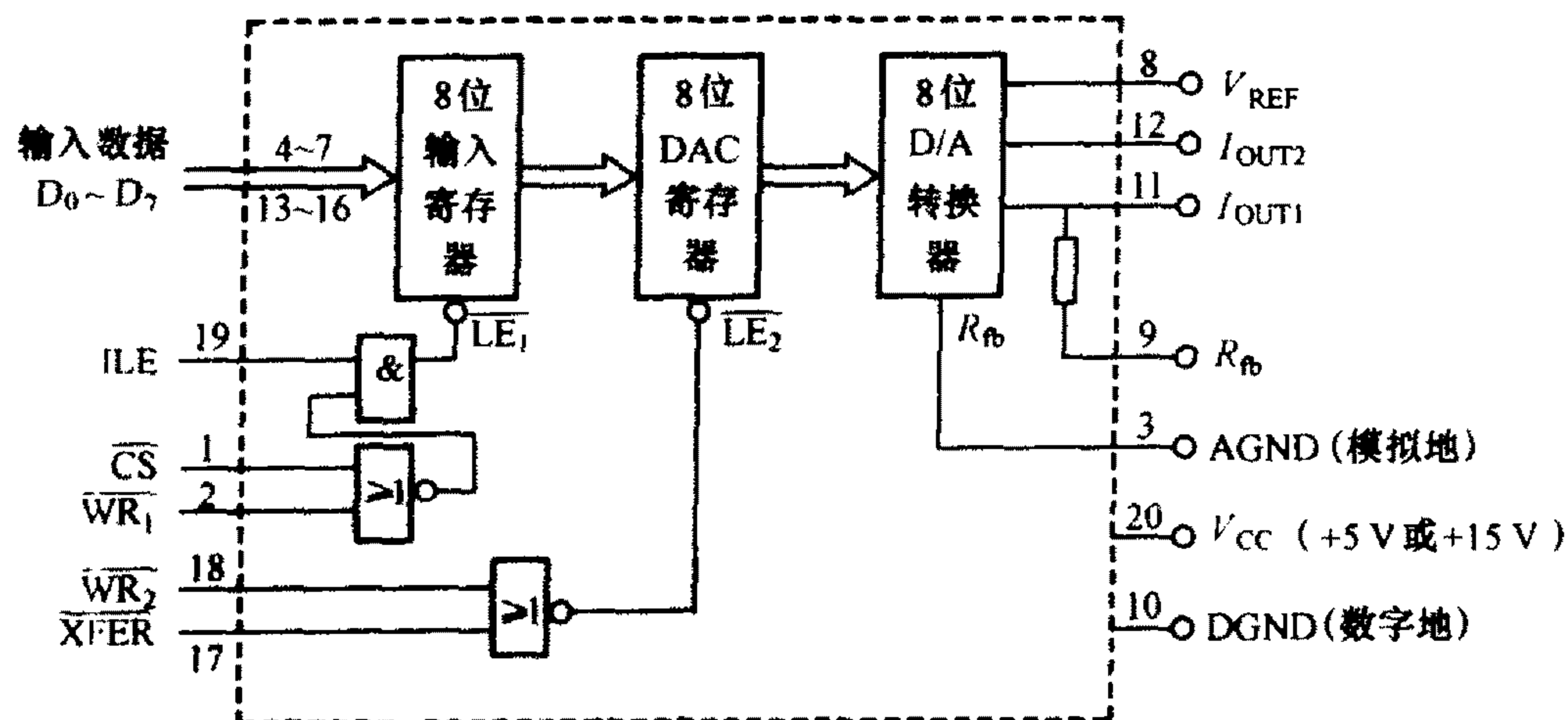


图 8.56 DAC0832 的内部结构示意图

I_{OUT2} 模拟电流输出端, DAC0832 为差动电流输出, 一般情况下 $I_{OUT1} + I_{OUT2} = \text{常数}$;

R_{fb} 反馈电阻引出端, 接运算放大器的输出;

V_{ref} 参考电压输入端, 要求其电压值要相当稳定, 一般在 $-10\text{ V} \sim +10\text{ V}$ 之间;

V_{CC} 芯片的电源电压, 可为 $+5\text{ V}$ 或 $+15\text{ V}$;

AGND 模拟信号地;

DGND 数字信号地。

2) DAC0832 的主要技术指标

- ① 分辨率 8 位;
- ② 线性误差 $(0.05\% \sim 0.2\%) \text{FSR}$ (满刻度);
- ③ 转换时间 $1 \mu\text{s}$;
- ④ 功耗 20 mW 。

3) 工作方式及线路连接

从图 8.56 可以看出, DAC0832 的内部包括两级锁存器: 第一级是 8 位的输入寄存器, 由控制信号 ILE 、 $\overline{\text{CS}}$ 和 $\overline{\text{WR}}_1$ 控制; 第二级是 8 位的 DAC 寄存器, 由控制信号 $\overline{\text{WR}}_2$ 和 $\overline{\text{XFER}}$ 控制。根据这两个锁存器使用方法的不同, DAC0832 有三种工作方式。

(1) 单缓冲工作方式

单缓冲工作方式是使输入寄存器或 DAC 寄存器中的任意一个工作在直通状态, 而另一个工作在受控锁存状态。例如, 要想使输入寄存器受控, DAC 寄存器直通, 则可将 $\overline{\text{WR}}_2$ 和 $\overline{\text{XFER}}$ 接数字地, ILE 接 $+5 \text{ V}$ 。此时, 将 $\overline{\text{CS}}$ 接端口地址译码器输出, $\overline{\text{WR}}_1$ 接 IOW 信号, 则当 CPU 向输入寄存器的端口地址发出写命令时 (即执行指令 $\text{OUT} < \text{输入寄存器端口地址} >$, $< \text{要转换的数据} >$), 数据就写入输入寄存器, 因为 DAC 寄存器为直通状态, 所以写入到数据寄存器的数据立刻进行数模转换。其电路连接如图 8.57 所示。

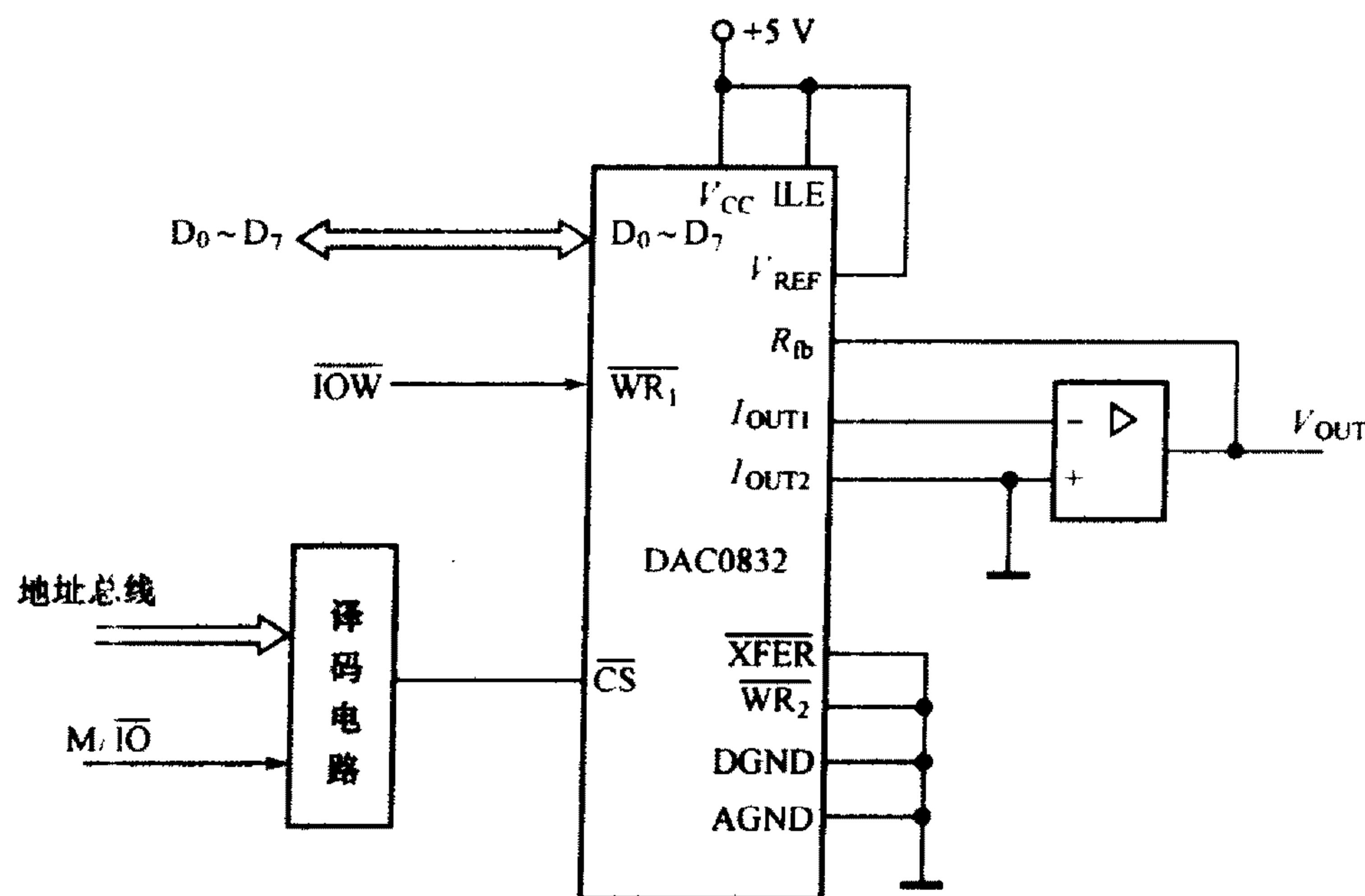


图 8.57 DAC0832 单缓冲方式下的电路连接

在只有单路模拟量输出通道, 或者虽有多路模拟输出通道但不要求同时刷新模拟输出时, 可采用这种方式。此种工作方式只用一条输出指令即可完成转换。下面是单缓冲工作方式的一个例子。

利用 DAC0832 实现 D/A 转换。DAC0832 工作在单缓冲方式。设 DAC0832 端口地址为 PORT, 待转换数据在 DATA 单元中。完成 D/A 转换的程序段如下:

```
MOV AL, DATA      ;要转换的数据送 AL
MOV DX, PORT       ;DAC0832 的端口地址送 DX
OUT DX, AL         ;将数字量送 D/A 转换器进行转换
```

(2) 双缓冲工作方式

在这种工作方式下, CPU 要对 DAC0832 进行两步写操作:

① 将数据写入输入寄存器;

② 将输入寄存器的内容写入 DAC 寄存器。具体过程为: 当 $\text{ILE} = 1, \overline{\text{CS}} = \overline{\text{WR}}_1 = 0$ 时, 待转换的数据被写入输入寄存器。随后 $\overline{\text{WR}}_1$ 由低电平变高电平, 数据出现在输入寄存器的输出端。在整个 $\overline{\text{WR}}_1$ 为高电平期间, 输入寄存器的输出端将不再随其输入端的变化而变化。从而保证了在数模转换时数据稳定不变。

锁存在输入寄存器中的数据此时并不能进入 DAC 寄存器, 只有当 $\overline{\text{XFER}} = \overline{\text{WR}}_2 = 0$ 时, 数据才能写入 DAC 寄存器, 并同时启动转换。双缓冲工作方式的工作时序如图 8.58 所示。其连接方法是: ILE 固定接 +5 V, $\overline{\text{WR}}_1$ 、 $\overline{\text{WR}}_2$ 均接到 $\overline{\text{IOW}}$, 而 $\overline{\text{CS}}$ 和 $\overline{\text{XFER}}$ 分别接到两个端口的地址译码信号线, 即 DAC0832 占用两个端口地址。

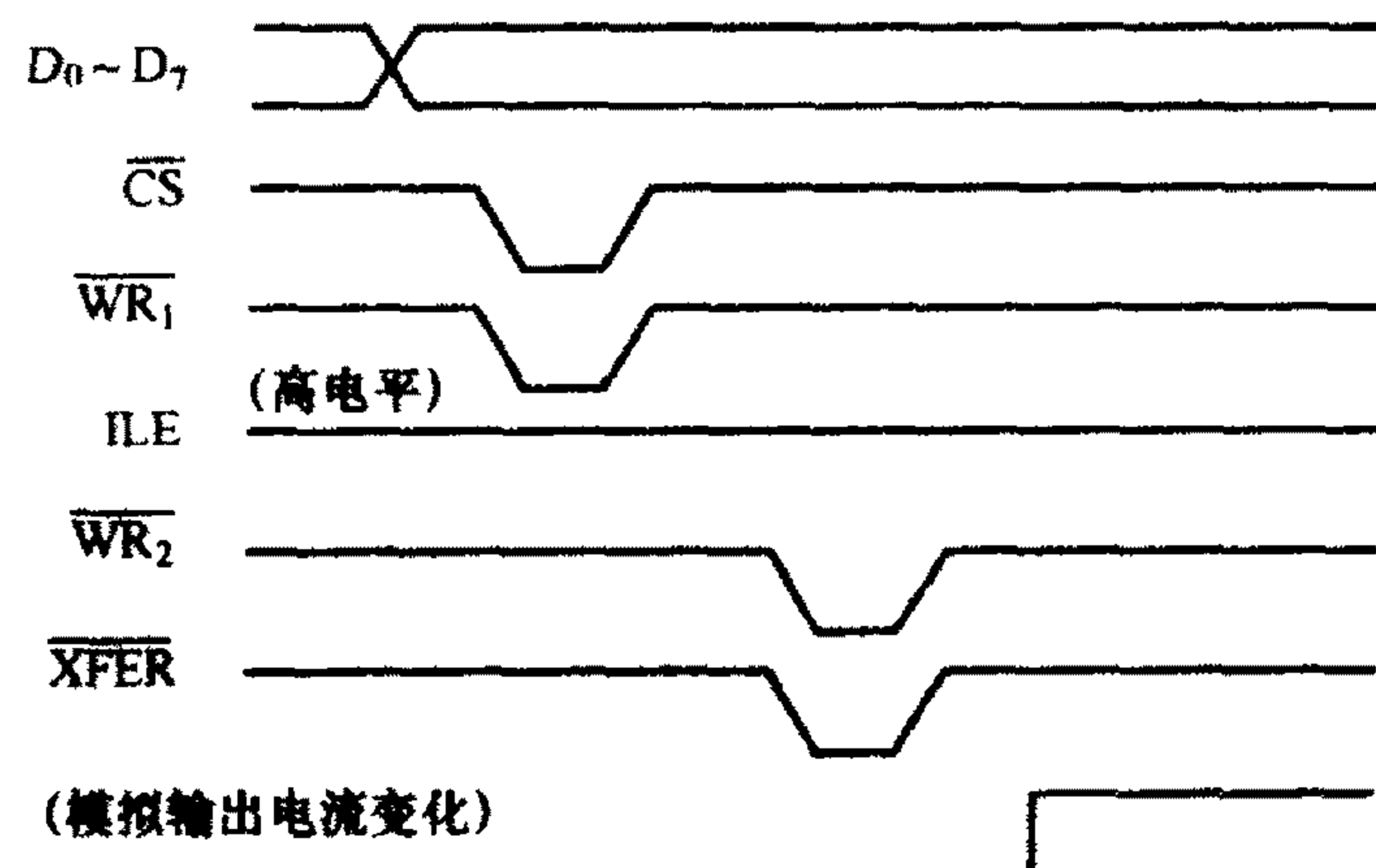


图 8.58 DAC0832 工作时序图

双缓冲工作方式的优点是数据接收和启动转换可以异步进行。可以在 D/A 转换的同时, 接收下一个数据, 提高了模数转换的速率。它还可用于多个通道同时进行 D/A 转换的场合, 其外部接线如图 8.59 所示。

由于在这种工作方式要求先使数据锁存到输入寄存器, 之后再使数据进入 DAC 寄存器进行数模转换, 所以, 在程序中需要安排两条 OUT 指令。双缓冲方式的程序段如下:

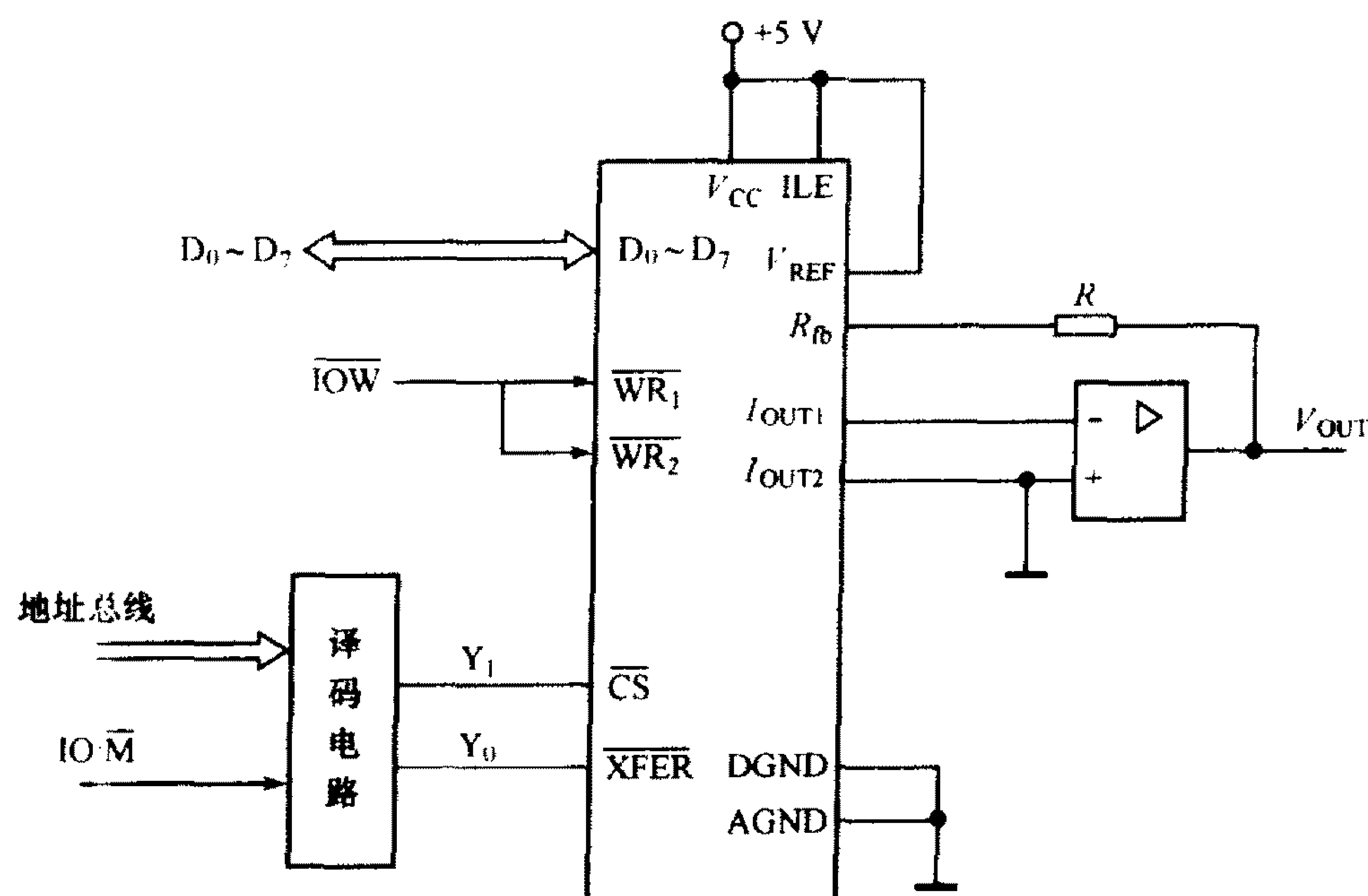


图 8.59 DAC0832 双缓冲方式下的电路连接

```

MOV AL, DATA
MOV DX, PORT1      ;输入寄存器端口地址送 DX
OUT DX, AL          ;数据送输入寄存器
MOV DX, PORT2      ;DAC 寄存器端口地址送 DX
OUT DX, AL          ;数据送 DAC 寄存器并启动转换
HLT

```

(3) 直通工作方式

这种工作方式是将 \overline{CS} 、 $\overline{WR_1}$ 、 $\overline{WR_2}$ 以及 \overline{XFER} 引出线都直接接数字地, ILE 接 +5 V, 芯片就处于直通状态。此时 DAC0832 就一直处于 D/A 转换状态, 即模拟输出端始终跟踪输入端 $D_0 \sim D_7$ 的变化。由于这种工作方式下 DAC0832 不能直接与 8088 CPU 的数据总线相连接, 故在实际工程中很少采用。

4) DAC0832 的应用

(1) 信号源

由上面的讨论可知, DAC0832 在单缓冲方式下可以直接与系统总线相连, 亦即可以将它看做一个输出端口。每向该端口送一个 8 位数据, 其输出端就会有相应的输出电压。可以通过编写程序, 利用 D/A 转换器产生各种不同的输出波形, 如锯齿波、三角波、方波、正弦波等。下面是根据图 8.60 所示的电路编写的一个输出任意周期锯齿波的程序。DAC0832 工作在单缓冲方式, 端口地址为 0278H。

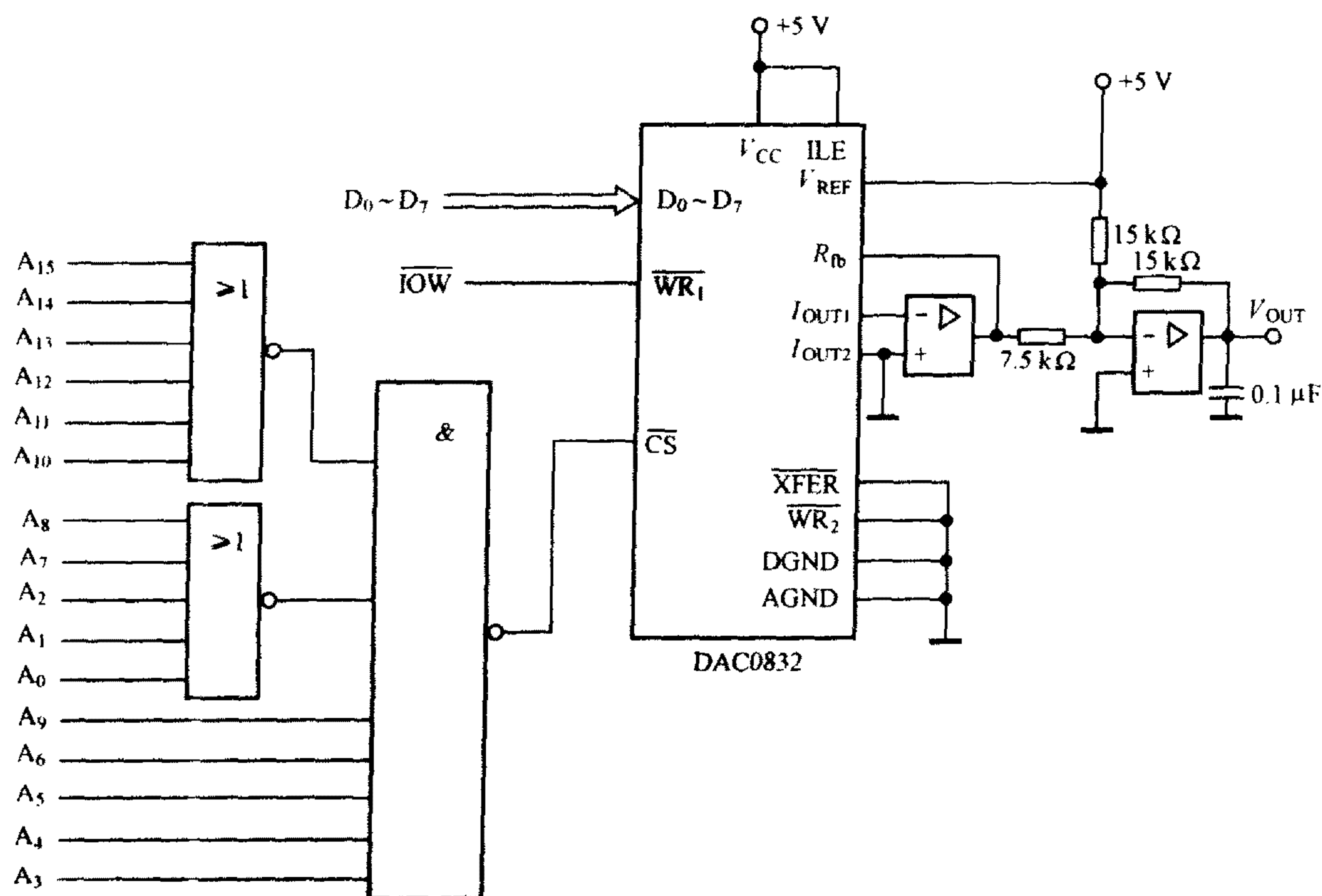


图 8.60 DAC0832 应用连接图

正向锯齿波的规律是电压从最小值开始逐渐上升,上升到最大值时立刻跳变为最小值,如此往复。(反向锯齿波正好相反,先从最小值跳变为最大值,然后逐渐下降到最小值。)所以只要从 0 开始往 DAC0832 输出数据,每次加 1,直到最大值 FFH,然后再从 0 开始下一个周期。这个过程循环执行即可在 DAC0832 输出端得到一个正向锯齿波。下面是一个产生反向锯齿波的程序段,这里使用了一个技巧,用 0 减 1 直接得到最大值 FFH,这样在锯齿波的齿根部可以少做一次判断:

```

MOV DX,0278H      ;端口地址送(DX)
MOV AL,0          ;初始值送(AL)
NEXT: OUT DX,AL    ;输出数字量到 D/A 转换器
DEC AL            ;数字量减 1
JMP NEXT          ;循环

```

注意,本程序产生的锯齿波不是平滑的波形,而是有 255 个小台阶,通过加滤波电路可以得到较平滑的锯齿波输出。还可以通过软件实现对输出波形周期和幅度的调整。

如果 DAC0832 输出电压范围为 0~5 V,但要求输出电压为 1~4 V 周期任意的正向锯齿

波,则相应的程序如下。因为已知当输出为 5 V 时,输入数字量为最大值 255,所以

$$1 \text{ V 电压对应的数字量} = 1 \times 255/5 = 51 = 33\text{H}$$

$$4 \text{ V 电压对应的数字量} = 4 \times 255/5 = 204 = 0\text{CCH}$$

程序段如下:

```

                MOV DX,0278H           ;0832 的端口地址送(DX)
NEXT1:          MOV AL,33H             ;最低输出电压对应的数字量送(AL)
NEXT2:          OUT DX,AL              ;输出数字量到 DAC0832
                INC AL                  ;数字量加 1
                CALL DELAY              ;调用延时子程序
                CMP AL,0CCH             ;到最大值(输出 4 V 电压)否
                JNA NEXT2               ;若没有到最大值继续输出
                JMP NEXT1               ;达到最大输出,则重新开始下一个周期
DELAY:          MOV CX,100              ;延时,延时常数可修改
DELAY1:          LOOP DELAY1
                HLT

```

本例中,不仅实现了波形幅度的调整,通过在延时子程序中设置不同的延时常数,还可以实现输出信号周期的调整。

(2) 工业控制器

D/A 转换器也常用于调速系统和伺服控制系统中的电机转速控制。图 8.61 给出了一个直流伺服电机的脉宽调制(PWM)转速控制系统。CPU 发出的控制信号经锁存器到 D/A 转换器,转换后的模拟电压通过功率放大器,控制直流伺服电动机的转速。如果再使用速度

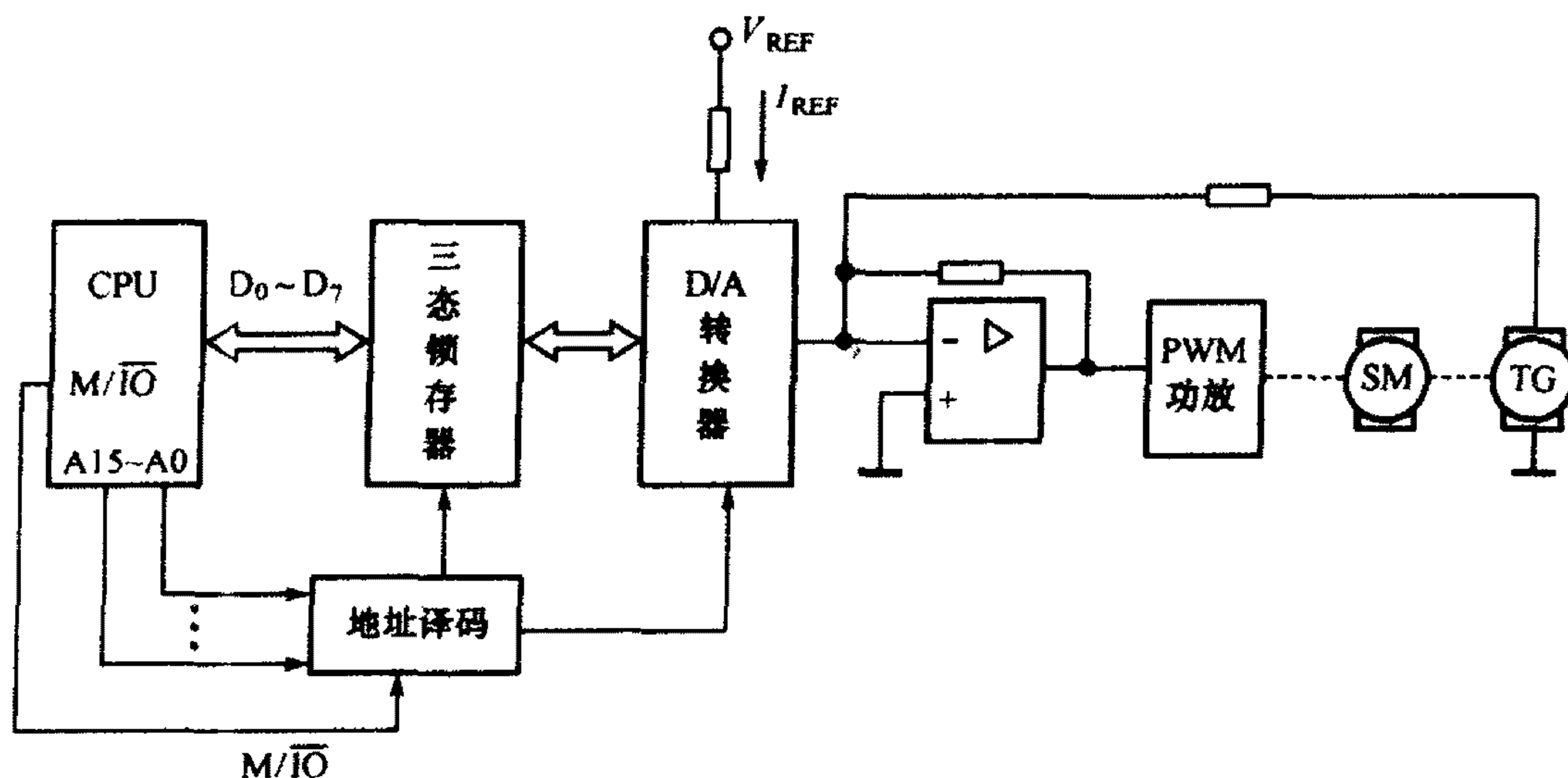


图 8.61 D/A 转换器在直流电机调速系统中的应用

传感器(如光电编码器等)将检测到的转速通过模拟量的输入通道反馈给微型计算机,即可构成一个闭环控制系统,可以达到很高的控制精度。

8.3.3 模数(A/D)转换器

A/D 转换器是将连续变化的模拟信号转换为数字信号,以便于计算机进行处理。它与 D/A 转换器一样,是微型计算机应用系统的一种重要接口,常用于数据采集系统。

A/D 转换器的种类很多,如计数型 A/D 转换器、双积分型 A/D 转换器、逐位反馈型 A/D 转换器等。考虑到精度及转换速度的折衷,这里以常用的逐位反馈型(也称逐位逼近型) A/D 转换器为例,来说明 A/D 转换器的一般工作原理。

1. A/D 转换器的工作原理及技术指标

1) A/D 转换器的工作原理

图 8.62 所示为逐位反馈型 A/D 转换器的内部结构,主要由逐次逼近寄存器 SAR、D/A 转换器、电压比较器和一些时序控制逻辑电路等组成。

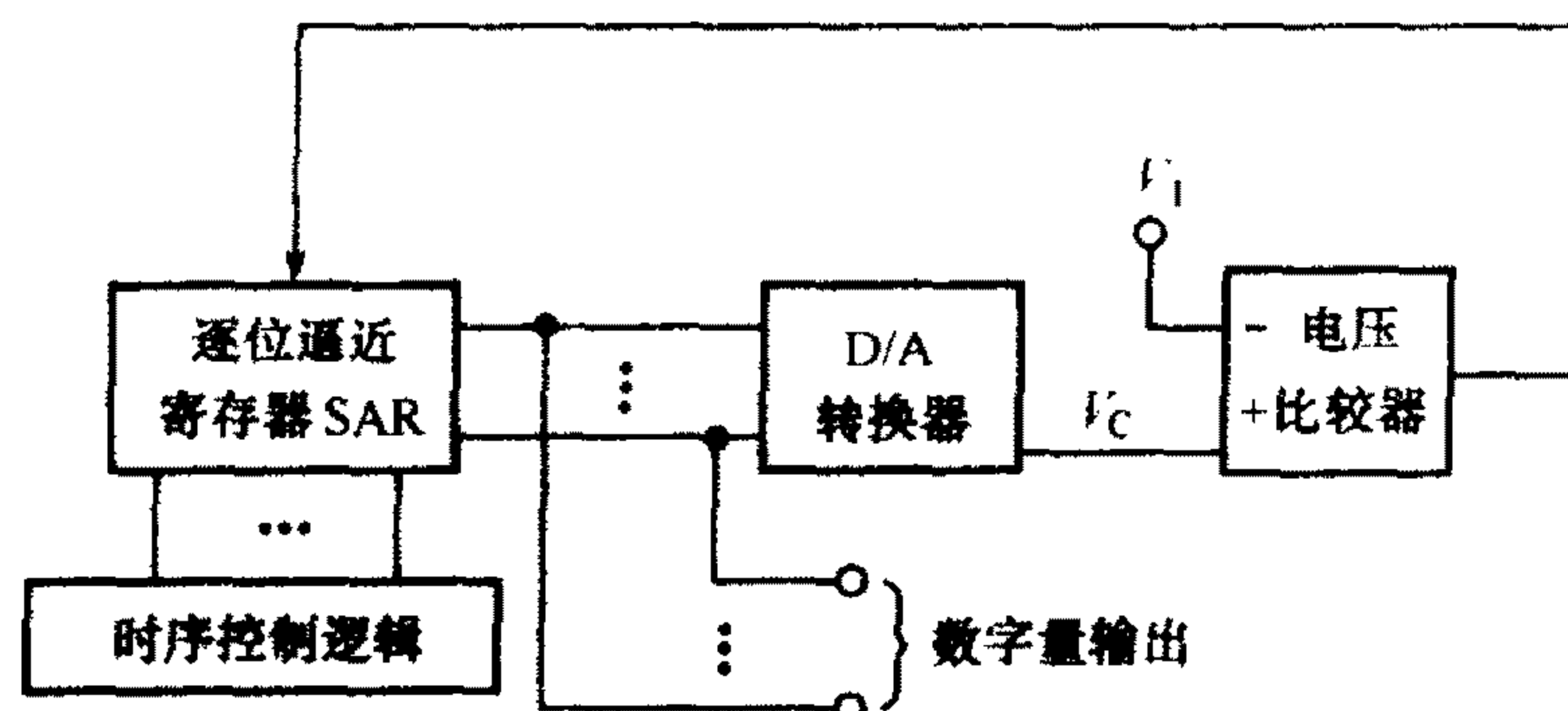


图 8.62 逐次反馈型 A/D 转换器的结构

逐位反馈型 A/D 转换器的工作原理非常类似于用天平称重。在转换开始前,先将 SAR 寄存器各位清零,然后设其最高位为 1(对 8 位来讲,即为 10000000B)——就像天平称重时先放上一个最重的砝码一样,SAR 中的数字量经 D/A 转换器转换为相应的模拟电压 V_C ,并与模拟输入电压 V_i 进行比较,若 $V_i \geq V_C$,则 SAR 寄存器中最高位的 1 保留,否则就将最高位清零——若砝码比物体轻,就要保留此砝码,否则去掉此砝码。然后再使次高位置 1,进行相同的过程……直到 SAR 的所有位都被确定。转换过程结束后,SAR 寄存器中的二进制码就是 A/D 转换器的输出。

2) A/D 转换器的主要技术指标

(1) 精度

A/D 转换器的转换精度由各种因素引起的误差所共同决定。这些误差有:

① 量化误差

A/D 转换器的量化误差(也称分辨率)决定于 A/D 转换器的转换特性。例如,一个 3 位的 A/D 转换器的转换特性如图 8.63 所示。当模拟量的值在 0~0.5 V 范围变化时,数字量输出为 000B;在 0.5 V~1.5 V 范围变化时,数字量输出为 001B。这样在给定数字量情况下,实际模拟量与理论模拟量之差最大为 ± 0.5 V。这种误差是由转换特性造成的,是一种原理性误差,也是无法消除的误差。从图中可以发现,数字量的每个变化间隔为 1 V,就是说模拟量在这个间隔内的电压变化,不会使数字量发生变化。

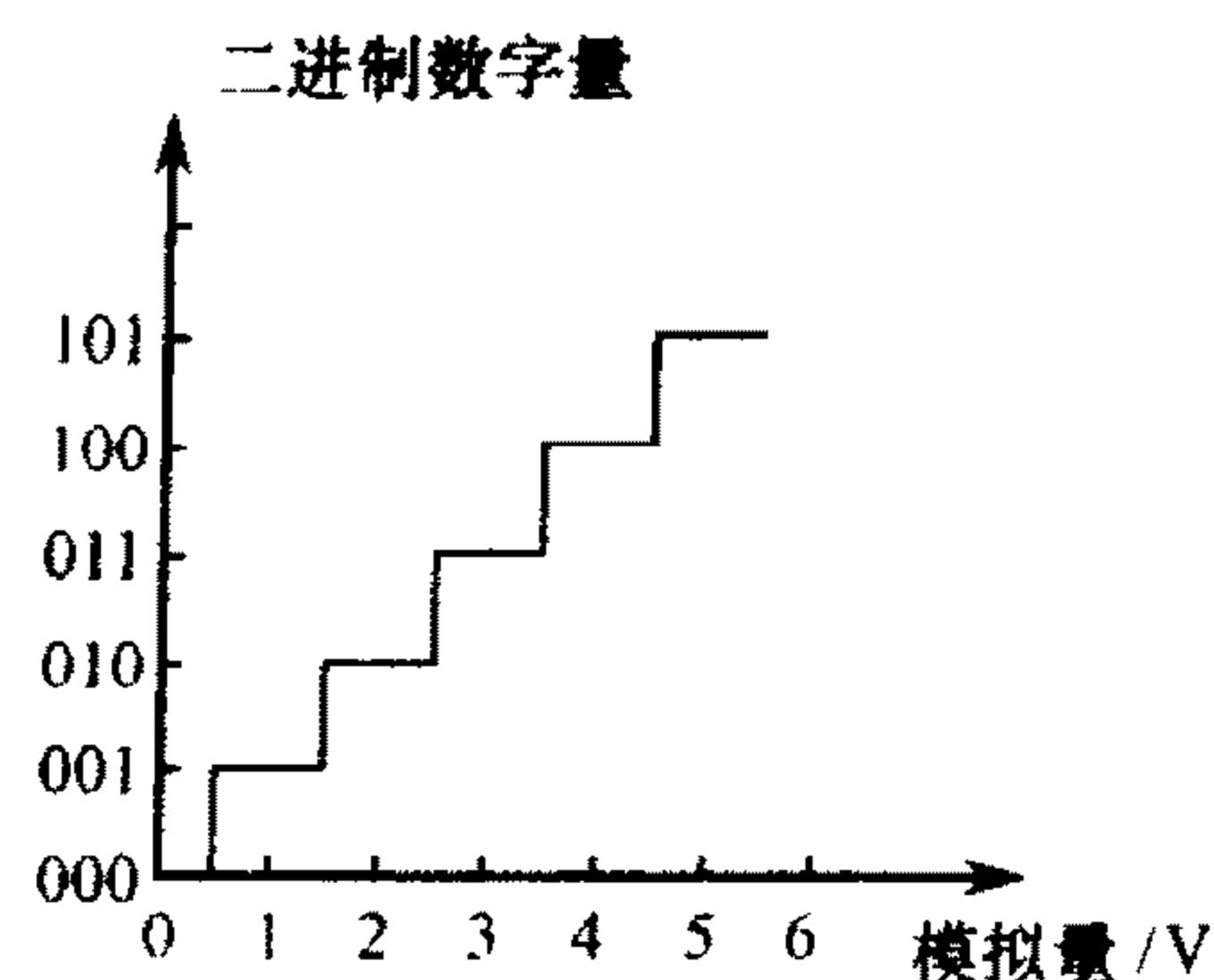


图 8.63 A/D 转换器的转换特性

这个间隔称为量化间隔(也称为当量),用 Δ 表示,其定义为

$$\Delta = \frac{\text{输入满度电压值}}{\text{A/D 转换器的最大数字量输出}} \quad (9.6)$$

对输出为 n 位的 A/D 转换器,其量化间隔 Δ 可表示为

$$\Delta = \frac{V_{\max}}{2^n - 1} \quad (9.7)$$

例如,对上例中的 12 位 A/D 转换器,若最大输入模拟电压为 5 V,则其量化间隔 Δ 为

$$\Delta = \frac{5 \text{ V}}{4095} \approx 1.22 \text{ mV} \quad (9.8)$$

而量化误差用绝对误差就可表示为

$$\text{量化误差} = \frac{1}{2} \times \text{量化间隔} = \frac{V_{\max}}{2(2^n - 1)} \quad (9.9)$$

也可用 $(1/2)\text{LSB}$ 来表示。

因此,一旦 A/D 转换器的位数确定,其量化误差也就确定了。

② 非线性误差

A/D 转换器的非线性误差是指在整個转换量程范围内,数字量所对应的模拟输入信号的实际值与理论值之间的最大差值。理论上 A/D 转换曲线应该是一条直线,即模拟输入与数字量输出之间应该是线性关系;但实际上它们两者的关系并非呈线性。所谓非线性误差就是由于二者关系的非线性而偏离理想直线的最大值。常用 LSB 来表示。

③ 其他误差

影响 A/D 转换器转换精度的因素还有:电源波动引起的误差,温度漂移误差,零点漂移误差,参考电源误差,等等。

(2) 转换时间

转换时间是指完成一次 A/D 转换所需要的时间,即从发出启动转换命令信号到转换结束信号有效之间的时间间隔。转换时间的倒数称为转换速率(频率)。例如 AD574KD 的转换时间为 $35\ \mu\text{s}$,则其转换速率为 $28.57\ \text{kHz}$ 。

(3) 输入动态范围

输入动态范围也称量程,指能够转换的模拟输入电压的变化范围。A/D 转换器的模拟电压输入分为单极性和双极性两种:

单极性:动态范围为 $0 \sim +5\ \text{V}$ 、 $0 \sim +10\ \text{V}$ 或 $0 \sim +20\ \text{V}$;

双极性:动态范围为 $-5\ \text{V} \sim +5\ \text{V}$ 或 $-10 \sim +10\ \text{V}$ 。

2. 典型的 A/D 转换器芯片及其应用

A/D 转换器芯片的种类很多。下面以较为常用的 ADC0809 为例,介绍它们与微型计算机系统的连接方法。

ADC0809 是逐位逼近型 8 位单片 A/D 转换芯片。片内含 8 路模拟开关,可允许 8 个模拟量输入。片内带有三态输出缓冲器,因此可直接与系统总线相连。它的转换精度和转换时间都不是很高,但其性能价格比有较明显的优势,是目前应用较为广泛的芯片之一。

1) ADC0809 的引出线及内部结构

ADC0809 的外部引出线排列如图 8.64 所示。共有 28 根引出线,其含义如下:

$D_0 \sim D_7$ 输出数据线;

$IN_0 \sim IN_7$ 8 路模拟电压输入端,可连接 8 路模拟量输入;

ADDA、ADDB、ADDC 通道地址选择,用于选择 8 路中的一路输入,ADDA 为最低位,ADDC 为最高位;

START 启动信号输入端,下降沿有效,在启动信号的下降沿,启动转换;

ALE 通道地址锁存信号,用来锁存 ADDA ~ ADDC 端的地址输入,上升沿有效;

EOC 转换结束状态信号,当该引出线输出低电平时,表示正在转换,输出高电平,则表示一次转换已结束;

OE 读允许信号,高电平有效,在其有效期间,CPU 将转换后的数字量读入;

CLK 时钟输入端;

$V_{\text{REF}}(+)$ 、 $V_{\text{REF}}(-)$ 参考电压输入端;

V_{CC} 5 V 电源输入;

GND 地线。

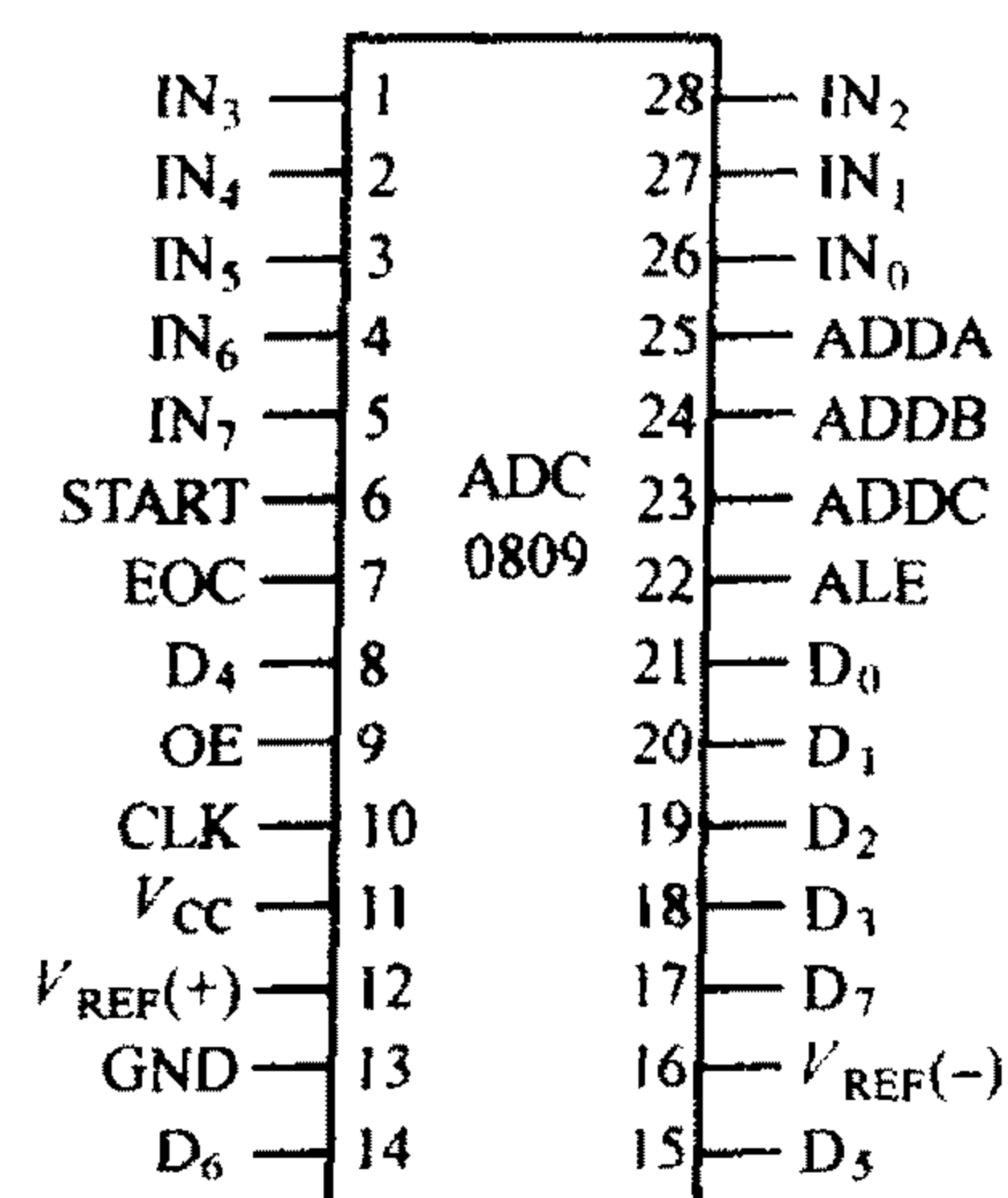


图 8.64 ADC0809 外部引出线排列图

ADC0809 需要外接参考电源和时钟。外接时钟频率为 10 kHz ~ 1.2 MHz。

ADC0809 的内部结构如图 8.65 所示,它主要由三部分组成:

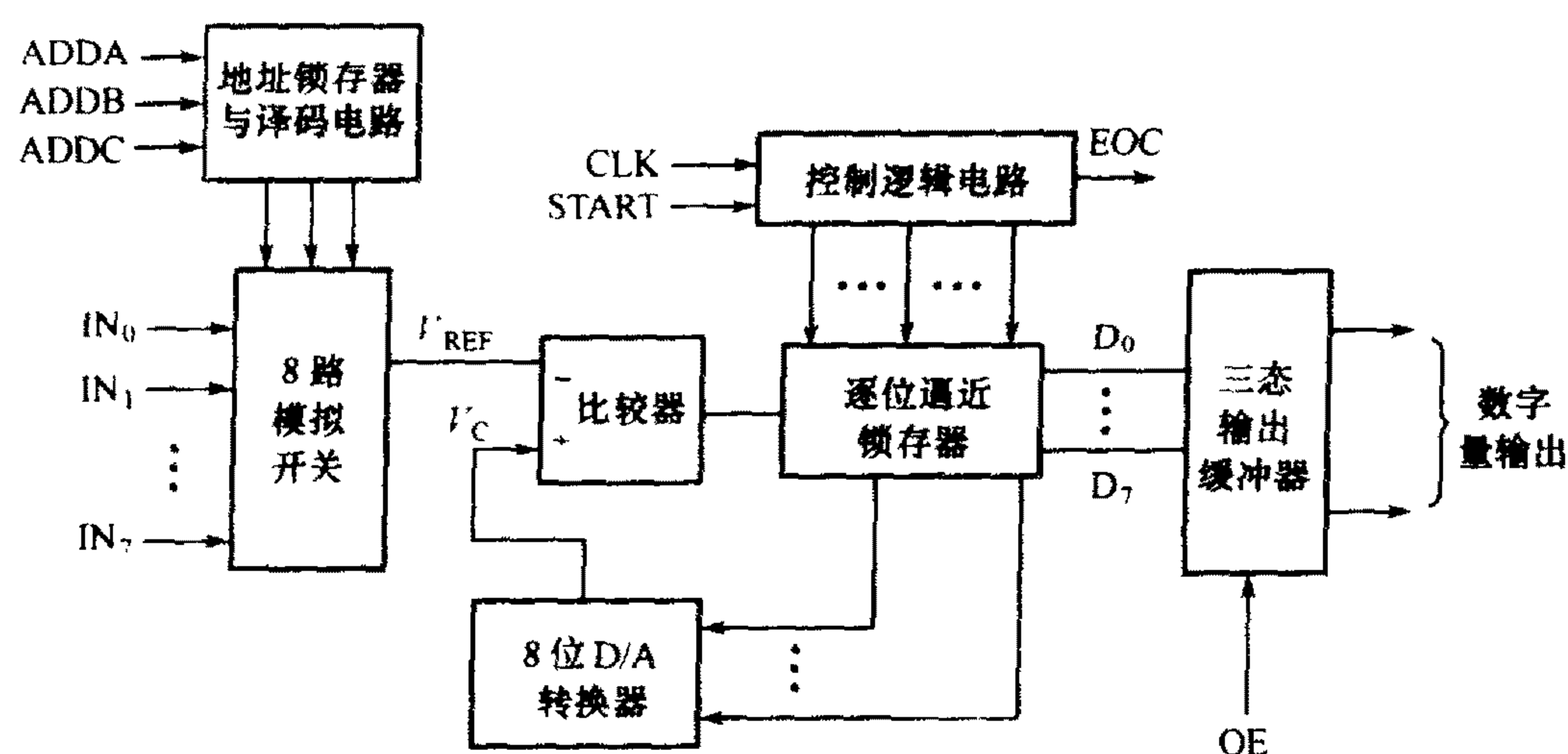


图 8.65 ADC0809 内部结构框图

① 模拟输入选择部分 包括一个 8 路模拟开关和地址锁存与译码电路。输入的三位通道地址信号由锁存器锁存,经译码电路译码后控制模拟开关选择相应的模拟输入。地址编码与输入通道的关系见表 8-9。

② 转换器部分 主要包括比较器、8 位 D/A 转换器、逐位逼近寄存器以及控制逻辑电路等。

③ 输出部分 包括一个 8 位三态输出缓冲器。

表 8-9 模拟通道选择

对应模拟通道	ADDC	ADDB	ADDA
IN ₀	0	0	0
IN ₁	0	0	1
IN ₂	0	1	0
IN ₃	0	1	1
IN ₄	1	0	0
IN ₅	1	0	1
IN ₆	1	1	0
IN ₇	1	1	1

2) ADC0809 的工作过程

ADC0809 的工作时序如图 8.66 所示。外部时钟信号通过 CLK 端进入其内部控制逻辑电路,作为转换时的时间基准。由时序图可以看出 ADC0809 的工作过程如下:

- ① 首先 CPU 发出 3 位通道地址信号 ADDC、ADDB、ADDA;
- ② 在通道地址信号有效期间,使 ALE 引出线上产生一个由低到高的电平变化,即脉冲上跳沿,它将输入的 3 位通道地址锁存到内部地址锁存器;
- ③ 接着给 START 引出线加上一个由高到低变化的电平,启动 A/D 转换;
- ④ 转换开始后,EOC 引出线呈现低电平,一旦转换结束,EOC 又重新变为高电平;
- ⑤ CPU 在检测到 EOC 变高电平后,输出一个正脉冲到 OE 端,将转换结果取走。

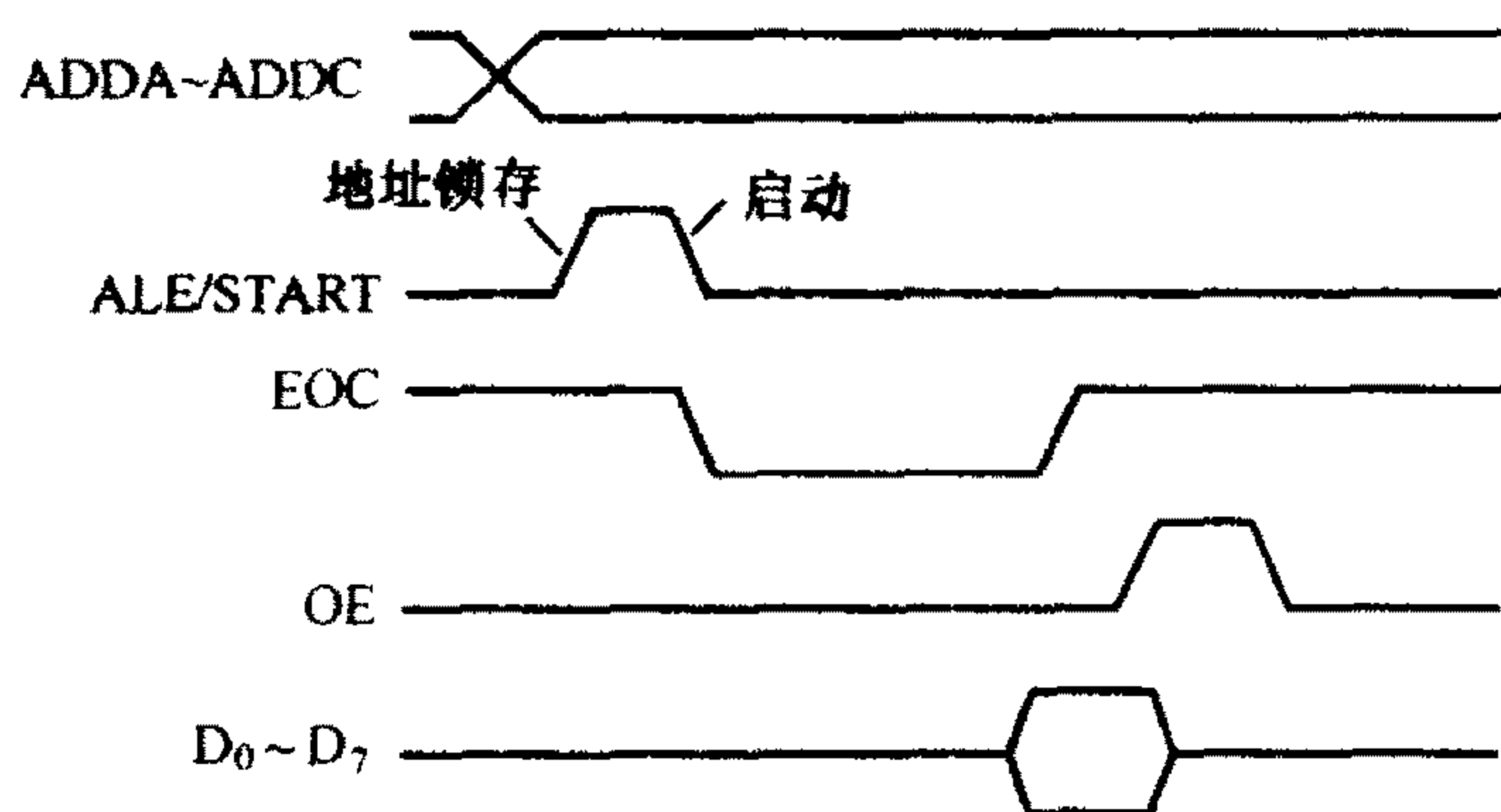


图 8.66 ADC0809 工作时序

3) ADC0809 的主要技术指标:

- ① 分辨率 8 位;
- ② 转换时间 100 μs ;
- ③ 电源 单电源 0 ~ +5 V。

4) ADC0809 与系统的连接方法

① 模拟信号输入 模拟信号分别连接到 $\text{IN}_0 \sim \text{IN}_7$ 端。当前要转换哪一路通过 ADDC ~ ADDA 的不同编码来选择。ADC0809 内部包括有地址锁存器,CPU 可通过一个输出接口(如 74LS273、74LS373、8255 等)把通道地址编码送到通道地址信号端。

② 数据线 $\text{D}_7 \sim \text{D}_0$ 的连接 由图 8.65 可知,ADC0809 的 $\text{D}_7 \sim \text{D}_0$ 输出端带有三态缓冲器,所以它可以直接连接到系统数据总线上。但考虑到驱动及隔离的因素,通常总是用一个输入接口与系统连接。

③ 启动转换信号的连接 ADC0809 采用脉冲启动方式。通常将 START 和 ALE 连接在一起作为一个端口看待。因为 ALE 是上升沿有效,而 START 是下降沿有效,这样连接就可用一个正脉冲来完成通道地址锁存和启动转换两项工作。初始状态下使该端口为低电平。

当通道地址信号输出后, CPU 往该端口送出一个正脉冲, 其上升沿锁存地址, 下降沿启动转换。

④ 状态信号 EOC 端的连接 判断一次 A/D 转换是否结束有几种方法:

用延时的方法: 编写延时程序, 使延时时间 \geq A/D 转换时间, 延时时间到, 读取转换结果。一般来说, 这种方式的实时性要差一些。

采用查询方式: 转换过程中, CPU 通过程序不断地读取 EOC 端的状态, 在读到其状态为“1”时, 则表示一次转换结束。

采用中断方式: 可将 ADC0809 的 EOC 端接到中断控制器 8259 的中断请求输入端, 当 EOC 端由低电平变为高电平时(转换结束), 即产生中断请求。CPU 在收到该中断请求信号后, 读取转换结果。由于 A/D 转换的过程需要一定的时间, 所以采用中断控制方式时 CPU 效率最高。

在将 ADC0809 连接到系统时, 为尽量少占用地址资源, 通常使 ADC0809 的各控制信号和输出端都通过输入/输出接口与系统相连。这里的 I/O 接口既可以是简单接口电路, 如 74LS244(做输入口)、74LS273(做输出口)等, 也可以采用前面介绍的可编程并行接口 8255 芯片(如图 8.67 所示)。当然, 若使用 8255 做输入/输出接口, 必须首先给 8255 初始化。

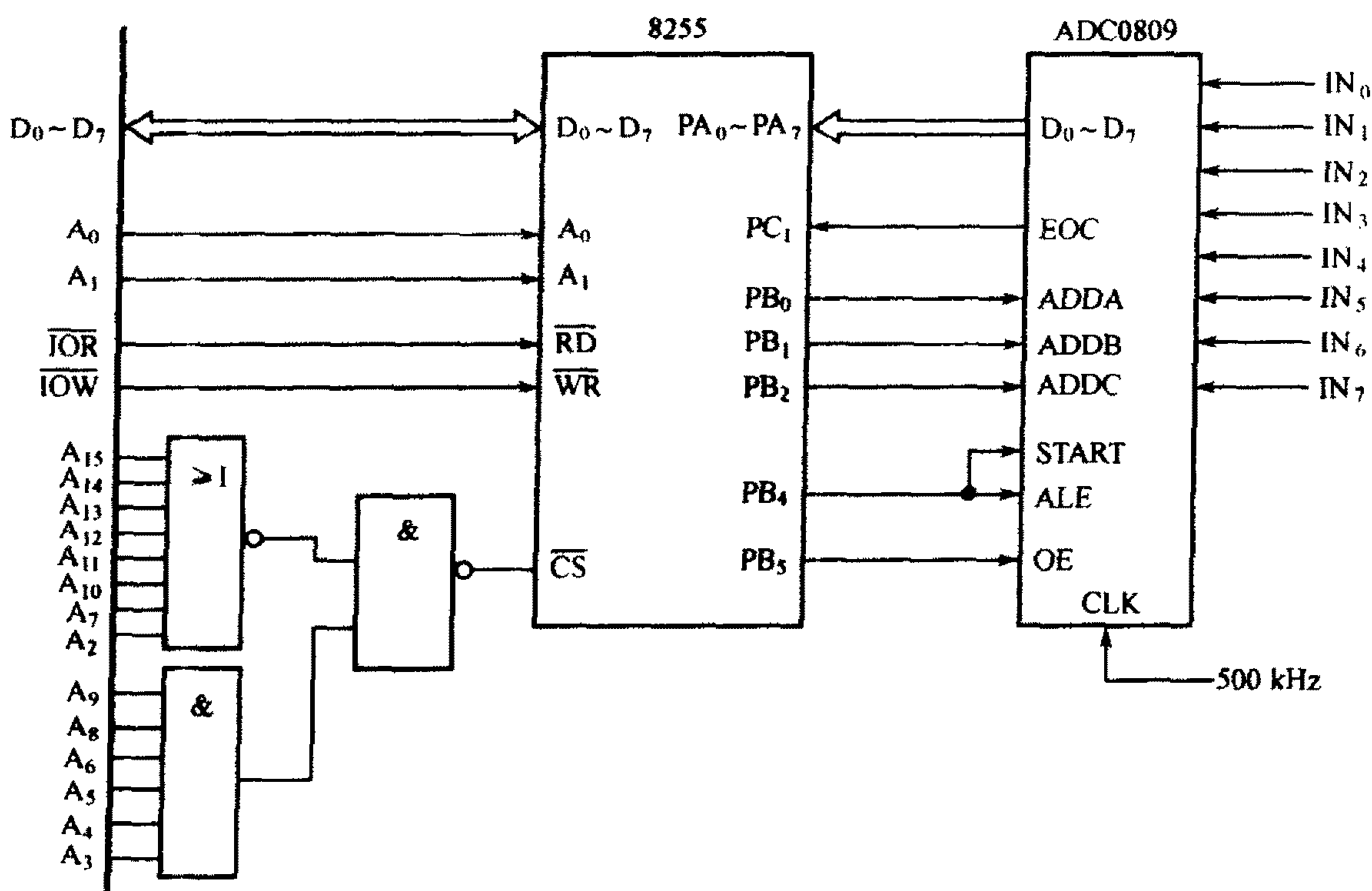


图 8.67 ADC0809 与系统连接图

5) ADC0809 的应用

ADC0809 主要用于数据采集系统中, 可以实现对 8 路模拟输入信号的循环数据采集。

现以图 8.67 为例,编写 8 路模拟量的循环数据采集程序。设转换结果(数字量)放在 DATA 为首的内存单元中。

由图 8.67 可知,8255 的地址为 0378H ~ 037BH。A、B、C 三个端口均工作在方式 0, A 口作为输入口,输入转换后的结果; B 口作为输出口,用来输出通道地址、发出地址锁存信号和启动转换信号; C 口低 4 位为输入口,用来读取转换状态,高 4 位没有使用。

初始化程序如下:

```

;初始化 8255 子程序
INIT_8255    PROC NEAR
               MOV DX,37BH
               MOV AL,91H      ;A、B、C 均为方式 0, A 入, B 出, C 入
               OUT DX, AL
               RET
INIT_8255    ENDP
;数据采集程序
START:        MOV AX,SEG DATA
               MOV DS,AX
               MOV SI,OFFSET DATA
               CALL INIT_8255   ;初始化 8255
               MOV BL,0        ;通道号,初始指向第 0 路
               MOV CX,8        ;共采集 8 次,每路采集一次
AGAIN:        MOV AL,BL
               MOV DX,379H
               OUT DX,AL        ;送通道地址
               OR AL,10H
               OUT DX,AL        ;送 ALE 信号(上升沿)
               AND AL,0EFH
               OUT DX,AL        ;输出 START 信号(下降沿)
               NOP              ;空操作等待转换
               MOV DX,37AH
WAIT1:        IN AL,DX          ;读 EOC 状态
               AND AL,02H
               JZ WAIT1         ;若 EOC 为低电平,则等待
               MOV DX,379H
               MOV AL,BL
               OR AL,20H

```



```
OUT DX, AL          ;EOC 端为高电平,则输出读允许信号 OE = 1
MOV DX, 378H
IN AL, DX           ;读入转换结果
MOV [SI], AL        ;将转换的数字量送存储器
INC SI              ;修改指针
INC BL              ;修改通道地址值
LOOP AGAIN          ;若未采集完,则再采集下一路数据
MOV DX, 379H
MOV AL, 0
OUT DX, AL          ;若 8 路数据已采集完,则回到初始状态
HLT
```

每执行一次以上 8 路模拟量的数据采集程序,内存数据段中以 DATA 为首地址的存储区域中就会存放 $IN_0 \sim IN_7$ 端模拟信号所对应的 8 位数字量。这个程序是通过查询 EOC 端口的状态来判断是否一次转换结束。若要用中断或延时的方法来决定是否转换结束,程序需要做一些修改,如何修改请读者自行考虑。

另外,在上述程序中是利用程序对读允许信号 OE 进行控制。实际上,也可将该端直接接到 +5 V 上,这样就可以将程序中对 OE 控制的指令删去。

以上通过典型的 A/D 转换器芯片 ADC0809,介绍了 A/D 转换器的工作原理、与系统的连接及其应用等,希望读者能够熟练地掌握它的使用方法,并由此在用到类似芯片时也能较容易地熟悉它们。

8.3.4 A/D 转换器和 D/A 转换器的综合应用实例

本节给出了一个使用 ADC0809 和 DAC0832 来采集和重放音频信号的实例。ADC0809 用于采集音频信号,转换成数字声音,并将它存储在存储器中。DAC0832 用于把数字声音转换成音频信号,在扬声器中重放出来。

图 8.68 给出了声音采集和重放的电路。用于采集和重放音频信号的程序如下。该程序转换 5 秒的语音信号,然后重放 3 次。重复此进程直到系统被关闭。本例中,语音被采样和存储在首地址为 AUDIO 的存储器区域中。声音采样频率为每秒钟 4 000 次,虽然这个采样频率有点低,音质受到一些影响,但作为一个实验性的系统,它提供的语音录入和播放质量尚在可接受的范围内。若要获得更好音质的录音和重放质量,最好用硬件实现连续采样,并用中断的方法把转换的音频数据存入存储器。此外,在需要更优良音质时,选用 10 位或 12 位的 A/D 转换器也是非常必要的。本例中假定 ADC0809 的 I/O 端口地址为 0700H(状态端口)、0701H(启动端口)和 0702H(数据端口)。DAC0832 的 I/O 端口地址为 0704H。

语音采集、重放程序(延时子程序未列出):



MODEL SMALL

. DATA

BUFFER DB 5 * 4 * 1024 DUP(0) ;为语音数据保留 20 KB 的存储空间

. CODE

STARTUP

START: **CALL** **READ** ;采集 5 秒的声音数据

MOV CX, 3 ;重放3次

AGAIN: CALL WRITE

LOOP AGAIN

JMP START

采集子程序

READ PROC NEAR

```

        LEA    DI, BUFFER          ;设置缓冲区指针
        MOV    CX, 4000            ;每秒采集 4 000 次
READ1:   MOV    DX, 0701H          ;启动转换
        OUT    DX, AL
READ2:   MOV    DX, 0700H          ;等待转换完毕
        IN     AL, DX
        TEST   AL, 1
        JZ     READ2              ;等待 EOC = 1
        MOV    DX, 0702H          ;从 ADC0809 读回声音数据
        IN     AL, DX
        MOV    [DI], AL           ;存入缓冲区
        INC    DI
        CALL   DELAY              ;延时 1/4 000 s
        LOOP   READ1              ;重复 4 000 次
        RET
READ     ENDP
;重放子程序
WRITE    PROC    NEAR
        PUSH    CX
        LEA     SI, BUFFER        ;设置缓冲区指针
        MOV     CX, 4000          ;共 4 000 个数据
WRITE1:  LODSB
        MOV     DX, 0704H          ;写声音数据到 DAC0832
        OUT     DX, AL
        CALL    DELAY              ;延时 1/4 000 s
        LOOP    WRITE1            ;重复 4 000 次
        POP     CX
        RET
WRITE    ENDP

```

习 题 八

8.1 设计一个能够驱动 8 个继电器的接口电路,要求接口能够锁存驱动信号。已知每个继电器的驱动电压为 12 V,电流为 200 mA。

8.2 设计一个 4 位七段数码显示器的接口电路。想一想,如何能简化多位七段数码显示器接口电路

的设计。

8.3 一般来讲,接口芯片的读/写信号应与系统总线上的哪些信号相连?

8.4 8253 可编程计数器有两种启动方式,在软件启动时,要使计数正常进行,GATE 端必须为什么电平?如果是硬件启动呢?

8.5 若 8253 芯片的接口地址为 0D0D0H~0D0D3H,时钟信号频率为 2 MHz。现利用计数器 0、1、2 分别产生周期为 10 μ s 的对称方波及每 1 ms 和 1s 产生一个负脉冲,试画出其与系统的电路连接图,并编写初始化程序。

8.6 某一计算机应用系统采用 8253 的计数器 0 做频率发生器,用计数器 1 产生 1 000 Hz 的连续方波信号,输入 8253 的时钟频率为 1.2 MHz。试问:初始化时送到计数器 0 和计数器 1 的计数初值分别为多少?计数器 1 工作于什么方式下?

8.7 用 8253 的计数器 2 产生连续脉冲信号,高电平时间为 100 μ s,低电平时间为 1 μ s。编写初始化程序,并说明计数器 2 的输入时钟频率是多少。

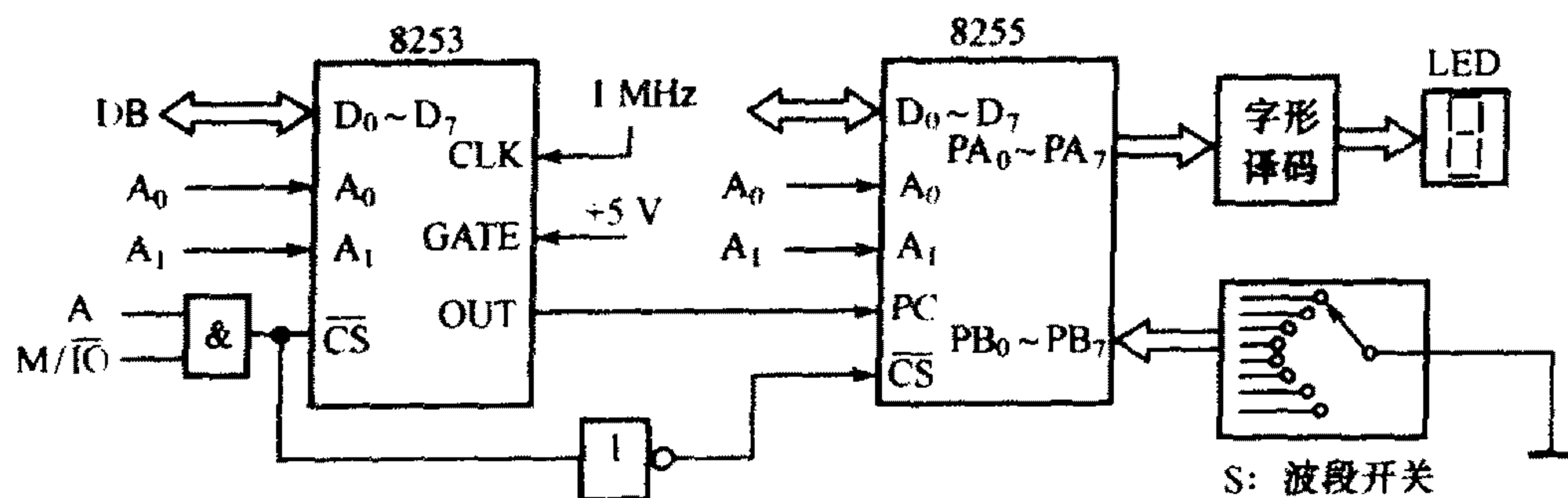
8.8 试比较并行通信与串行通信的特点。

8.9 8255 各端口可以工作在哪儿种方式下?当端口 A 工作在方式 2 时,端口 B 和 C 工作于什么方式下?

8.10 某 8255 芯片的地址范围为 0A380H~0A383H,工作于方式 0,A 口、B 口为输出口,现欲将 PC₄ 置 0,PC₇ 置 1,试编写初始化程序。

8.11 已知某 8088 微型计算机系统的 I/O 接口电路框图如下图所示。试完成:

- (1) 根据图中接线,写出 8255、8253 各端口的地址;
- (2) 编写 8255 和 8253 的初始化程序。其中,8253 的 OUT₁ 端输出 100 Hz 方波,8255 的 A 口为输出,B 口和 C 口为输入;
- (3) 为 8255 编写一个 I/O 控制子程序,其功能为:每调用一次,先检测 PC₀ 的状态,若 PC₀ = 0,则循环等待;若 PC₀ = 1,可从 PB 口读取当前开关 S 的位置(0~7),经转换计算从 A 口的 PA₀~PA₃ 输出该位置的二进制编码,供 LED 显示。



8.12 串行通信接口芯片 8250 的给定地址为 83A0H~83A7H,试画出其与 8088 系统总线的连接图。若采用查询方式由该 8250 发送当前数据段中偏移地址为 BUFFER 的顺序 100 个字节的数据,试编写发送程序。

8.13 若串行通信接口芯片 8250 有多个中断产生时,当前尚未被处理的中断会不会丢失?在中断处

理程序中应如何处理多个同时产生的中断？试画出中断服务程序的流程图。

8.14 试说明将一个工业生产现场的非电物理量转换为计算机能够识别的数字信号主要需经过哪几个过程。

8.15 对于一个 10 位的 D/A 转换器,其分辨率是多少？如果输出满刻度电压值为 5 V,其一个最低有效位对应的电压值等于多少？

8.16 某一测控系统要求计算机输出模拟控制信号的分辨率必须达到 1‰,则应选用的 D/A 转换芯片的位数至少是多少？

8.17 DAC0832 在逻辑上由哪几个部分组成？可以工作在哪儿种模式下？如果要求同时输出 3 路模拟量,则 3 片同时工作的 DAC0832 最好采用哪一种工作模式？

8.18 设 DAC0832 工作在单缓冲方式下,端口地址为 034BH,输出接运算放大器。试画出其与 8088 系统的线路连接图。

8.19 对 8 位、10 位和 12 位的 A/D 转换器,当满刻度输入电压为 5 V 时,其量化间隔各为多少？绝对量化误差又为多少？

8.20 某工业生产现场的 3 个不同点的压力信号经压力传感器、变送器及信号处理环节等分别送入 ADC0809 的 IN_0 、 IN_1 和 IN_2 端。要求用计算机巡回检测这 3 点的压力。现利用 8255 将 ADC0809 与系统相连,8255 的地址范围为 03F6H ~ 03F9H。试画线路连接图,并编写包括 8255 初始化程序在内的循环数据采集程序。

8.21 设被测温度的变化范围为 0°C ~ 100°C ,若要求测量误差不超过 0.1°C ,应选用分辨率为多少位的 A/D 转换器？

第9章 常用外设及设备驱动程序

本章主要介绍三方面的内容,一是计算机系统基本配置中的外部设备,包括它们的工作原理及结构特点;二是设备驱动程序的概念及 Windows 下的设备驱动程序;最后简要介绍计算机中的多媒体技术。

9.1 常用外部设备

人们通过输入设备将原始数据、被检测到的现场信息等送入计算机,计算机则将运算和处理的结果通过输出设备送出。我们将所有的输入/输出设备都统称为外部设备(简称外设),并通过各种外部设备与计算机进行信息交换。

常用的外设有键盘、显示器、打印机、鼠标等,它们属于计算机系统的基本配置,通过插在主板上的相应的接口卡与主机相连。这些设备以二进制、十六进制或 ASCII 码形式与计算机进行信息交换。另外,随着计算机应用领域的拓展,如网络和多媒体的应用,一些特殊外部设备,如网卡、调制解调器、扫描仪、语音合成器、图像捕捉设备、数字相机等也越来越普遍地得到使用。那些用于多媒体信息处理的外部设备可以按照人感官容易接受的图形、图像、语音等信息形式进行传输。多媒体形式的信息必须通过转换,形成计算机所能处理的形式,才可与计算机进行交互。

9.1.1 键盘

计算机的键盘系统包括键盘、键盘接口和键盘中断服务程序 3 个部分。PC 系列微型计算机中的键盘包括标准键盘(83 键、84 键)和扩展键盘(101 键、102 键)两种。

1. 键盘的工作原理

键盘内主要由单片机、译码器和 16 行 × 8 列的键开关阵列这三部分组成。所谓单片机,就是将主机的 4 个组成部分: CPU、存储器、总线及接口集成在一片硅片上。不同性能的单片机,这 4 部分的性能、容量等有较大的差别。键盘中使用的单片机通常是 8 位字长的,内含 2KB 的 ROM、128B 的 RAM、2 个 8 位 I/O 接口、1 个 8 位定时/计数器以及时钟发生器等。

键开关阵列是由单片机的输入/输出线及按键排列成的行、列结构,按键设置在行、列线

的交点上。这种结构的键盘也称为矩阵式键盘。在按键数量较多时,矩阵式键盘可以节省单片机的 I/O 接口线。 16×8 的行列结构可构成 128 个键的键盘。而实际上,大多数 PC 键盘的键数少于 100 个左右,因此在某些键位上并没有安装键开关。

行列式键盘的基本工作原理如图 9.1 所示。

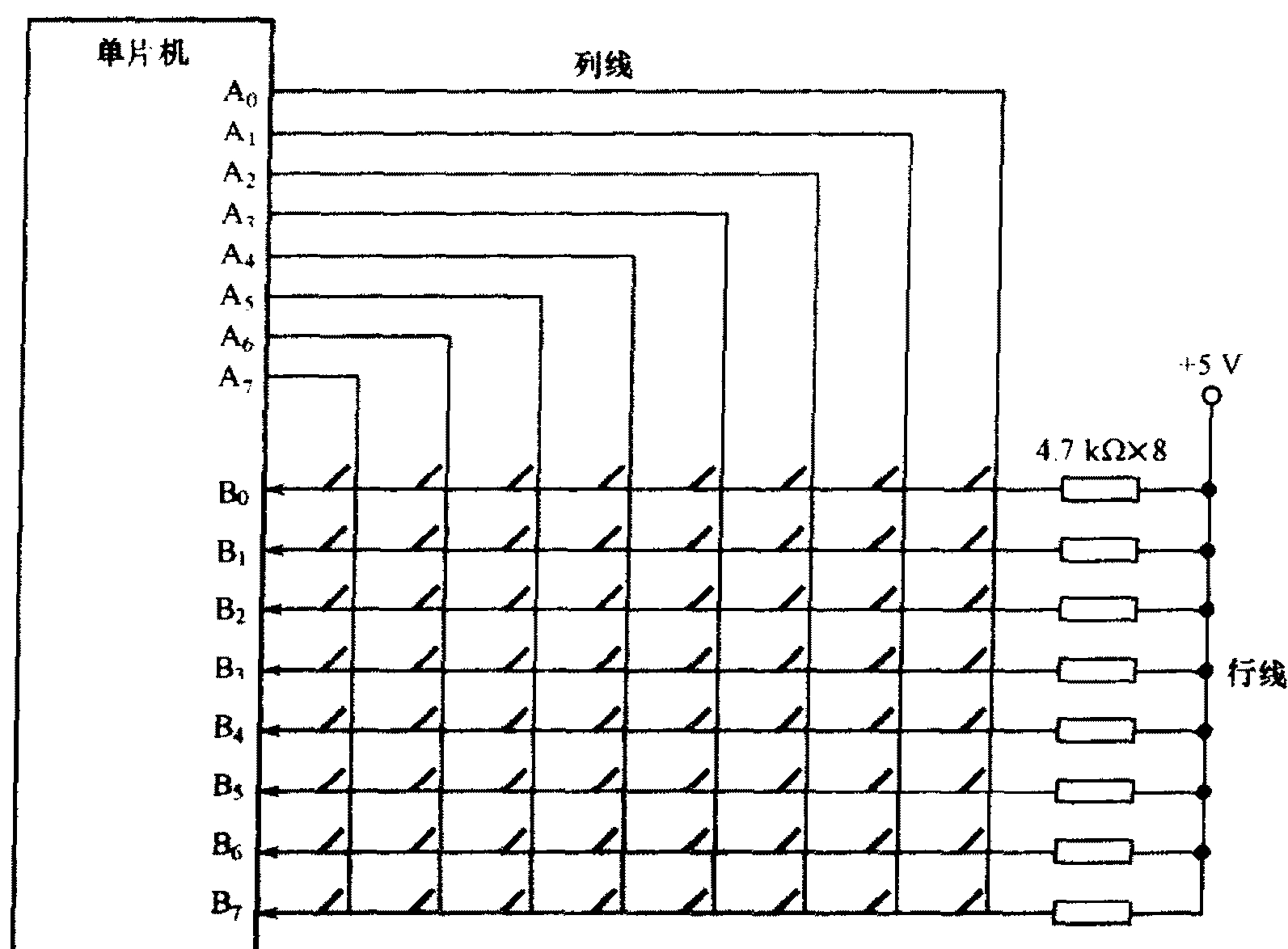


图 9.1 行列式键盘原理电路

键盘的按键设置在行、列线的交点上,行、列线分别连接到按键开关的两端。 $+5\text{ V}$ 电压通过上拉电阻接到行线上,使行线被钳位在高电平状态。

键盘工作时主要完成 3 项任务,一是键开关状态的可靠输入,二是判断有无键按下和哪一个键按下,三是把按下的键的键值送给计算机。

按键开关在开闭过程中不可避免地会出现瞬态抖动,其时间长短约为 $5 \sim 10\text{ ms}$ 。在抖动时检测键盘状态是不可靠的,因此要进行去抖动处理。去抖动可用硬件或软件实现。硬件去抖动电路通常由一个 R-S 触发器或单稳态电路构成。软件去抖动的方法是在检测到有键按下时,先延迟 10 ms 再检测键是否仍保持闭合状态。

键盘中哪一个键按下是由列线逐列置低电平后,检查行输入状态决定的。其方法是:依次给列线送低电平,然后检查所有行线状态,如果全为 1,则所按下的键不在此列。如果不全为 1,则按下的键必在此列,而且是在与处于 0 电平状态的行线相交点上的那个键。

以上这些键盘扫描、判断按键的操作由单片机来完成,除此之外,它还要完成将按键的键

号转换成键扫描码、对扫描码进行并串转换,并将串行的键扫描码和时钟送到主机等任务。

2. 键盘接口

微型计算机的键盘接口安装在系统主机板上,主要芯片是键盘接口控制器 8042。它通过一个五芯插头与键盘相连。图 9.2 给出了它们的硬件连接示意图。

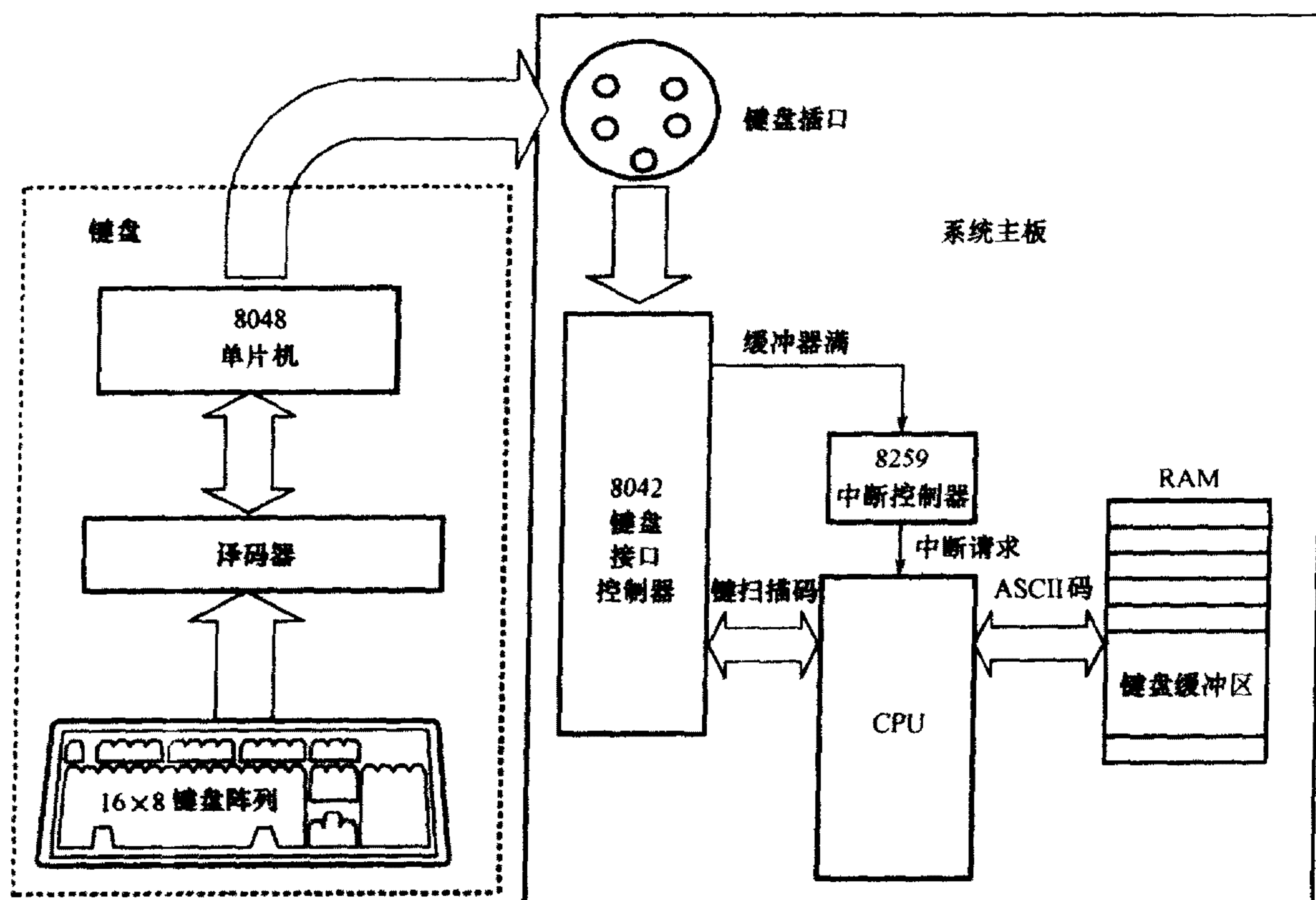


图 9.2 键盘接口的硬件连接示意图

8042 在逻辑上的简化框图如图 9.3 所示,4 个寄存器是从逻辑上等效画出的,其作用综合体现了 8042 的功能:

- ① 接收键盘送来的键盘扫描码。
- ② 输出缓冲器满时,产生键盘中断 当键盘扫描码送入输出缓冲区后,输出缓冲区满标志置位,产生键盘中断请求信号(IRQ_1)。当 CPU 响应 IRQ_1 后,就调用 BIOS 中的 9 号(INT_9)中断服务程序进行键盘代码转换处理,把扫描码转换成 ASCII 码和控制码,并存入 RAM 中的键盘缓冲区。

- ③ 接收并执行系统命令 用于对键盘进行初始化、测试、复位等操作。

3. 键盘与 CPU 的通信方式

键盘与计算机系统之间的通信以中断控制方式进行。中断类型包括一个类型码为 09H

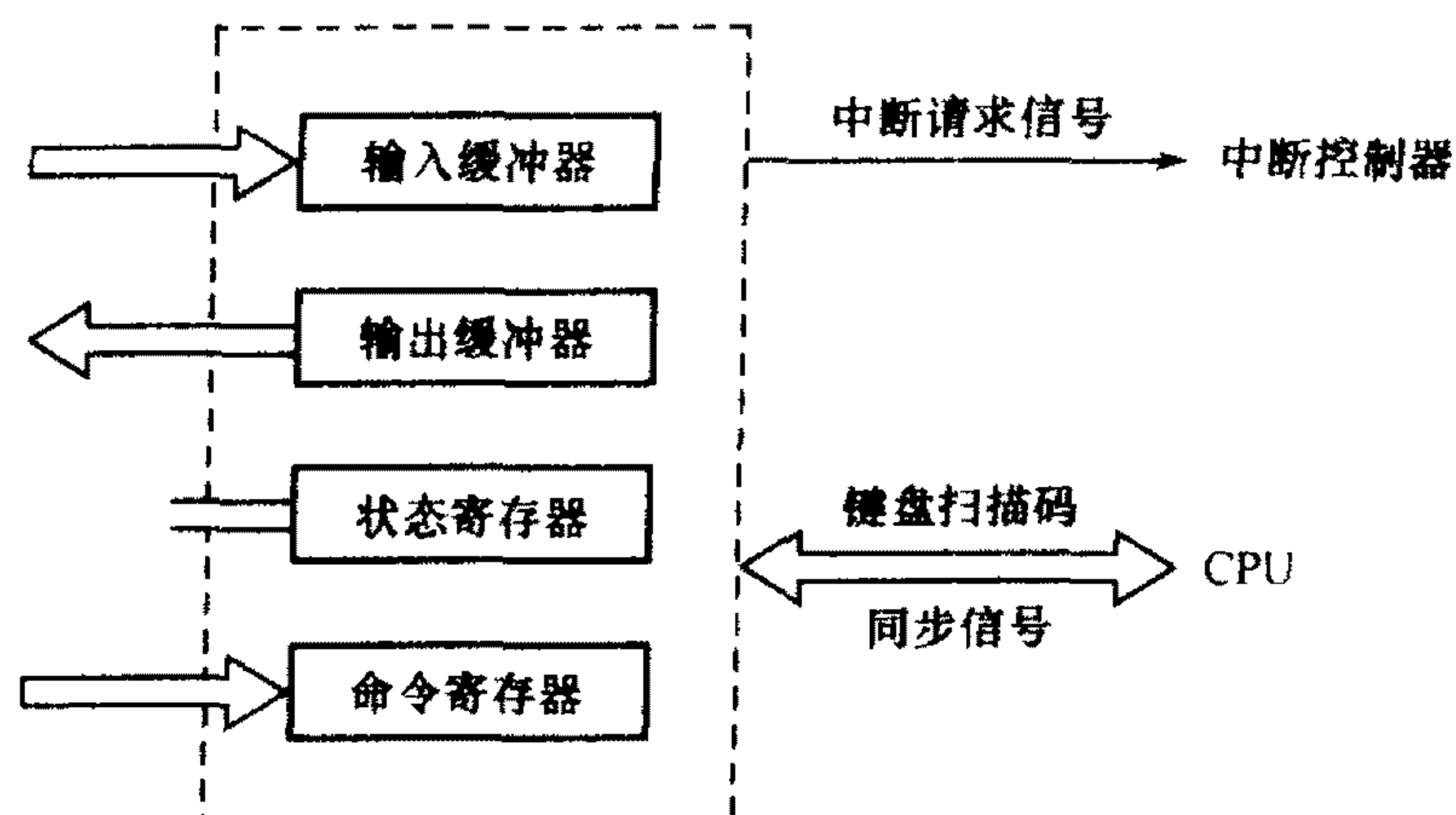


图 9.3 键盘接口控制器 8042 的简化逻辑框图

的外部硬件中断和 BIOS 子功能包中的类型码为 16H 的软中断。

1) 硬件中断

外部硬件中断是由按键动作引发的中断。其中断服务程序的主要功能是从键盘接口控制器 8042 中接收键盘按键信息,并在进行处理后将相应的结果存储在键盘缓冲区和键盘状态标志单元中。

键盘上共有 10 个特殊键,其键名和对应的扫描码见表 9-1。键盘的硬中断服务程序的重要任务之一就是判别从键盘来的是否为某个特殊键的扫描码。如果是,则首先根据当前键盘状态标志设置标志状态,以控制后续键代码的生成;如果不是特殊键,则程序就将其扫描码转换为两字节的 ASCII 码或扩展码,送到内存 BIOS 数据区中的键盘缓冲区。

表 9-1 特殊键及其扫描码

特殊键名	扫描码
Ctrl	1DH
Alt	38H
Shift - L	2AH
Shift - R	36H
Num Lock	45H
Scroll Lock	46H
Caps Lock	3AH
Ins	52H
Del	53H
Sys	54H

09H 号硬件中断共完成两种转换：

① 把键的扫描码转换为 ASCII 码。生成两个字节,低字节为 ASCII 码,高字节是系统的扫描码。

② 把键的扫描码转换为扩展码。同样生成两个字节,低字节为 0,高字节对应值为 0~255(通常功能键和某些组合键对应的是扩展码)。

2) BIOS 软中断

与 DOS 操作系统中包含有一个子功能包类似,ROM BIOS 中也包含了一组子功能包,它们同样是通过软中断指令 INT 来调用的。其中的一组子程序是关于键盘操作,其中断类型码为 16H,用于检查是否有键输入,并完成从键盘缓冲区取出键值的操作。16H 软中断共有 3 个子功能,见表 9-2。

以上两种各自独立的中断程序,都借助于键盘缓冲区来传递键值。

表 9-2 INT 16H 功能表

功能号	入口参数	出口参数	说明
0	AH = 0	AX 存放 ASCII 键或扩展码键符	从键盘读一个字符
1	AH = 1	ZF = 1, 无键符	检测输入字符是否准备好
2	AH = 2	ZF = 0, 有键符, 存在 AX 中 AL = KB_FLAG(键标志)	取当前特殊键的状态

3. 键盘缓冲区

键盘缓冲区是一个由 16 个字组成的按“先进先出”(FIFO)规则控制读/写操作的存储区,用于存放键的 ASCII 码。当 CPU 响应键盘中断时,在中断处理程序中将键的 ASCII 码写入缓冲区,供系统软件或应用软件在需要时读取。键盘缓冲区的作用可归结为以下两点:

① 实现键盘实时输入要求 实时处理用户随机键入的键码。

② 满足随机应用的需要 应用程序需要键盘输入的时刻不一定与按键同步。键盘缓冲区可协调键盘输入与程序运行之间的同步问题。此外,键盘缓冲区满足操作员快速键入的要求。

键盘缓冲区从逻辑上讲是一个循环队列,其队列形式由软件实现。进队列由 INT 09H 处理程序完成;出队列由 INT 16H 程序来完成。为此,循环队列需要设置头、尾两个指针。头指针(Head)总是指向缓冲区最早压入的键码位置;尾指针(Tail)总是指向最后压入的键码的下一个位置(注意,每个键码位置为 2B)。

队列操作如下:

① 进队列 键码进入尾指针所指向的单元。同时调整尾指针,指向下一单元。当尾指

针指向队列末端时,则返回到队列始端。

② 出队列 键码从头指针指向的单元中取出,同时调整头指针指向下一单元。当头指针指向队列末端时,则头指针返回队列始端。

③ 队列空 当头指针和尾指针相等时,表明队列已空,无键码可取。

④ 队列满 当尾指针修正为新的尾指针之后,它与头指针相等,表明队列已满,键码不能再存进键盘缓冲区。键盘缓冲区共占 32B,当尾指针加 2 等于头指针时,为队列满,故键盘缓冲区最多可存 15 个键码。

4. IBM PC 机键盘的特点和分类

1) 键盘特点

IBM PC 系列键盘具有两个基本特点:

① 按键开关均为无触点的电容开关,是通过按键的上下动作,使电容量发生变化来检测按键的断开或接通。除电容式开关外,常见的键开关还有霍尔效应式开关和触点式开关。

② PC 系列键盘属于非编码键盘。

2) 键盘分类

按照不同的功能,一般将键盘分为编码键盘和非编码键盘两种。

① 编码键盘 使用这种键盘时,当有键按下,系统可以自动检测,并能提供按键的对应键值。这种键盘接口简单,使用方便,但价格较高。

② 非编码键盘 这种键盘只简单提供键的行列位置(位置码或称扫描码),而按键的识别和键值的确定等工作全靠软件完成。

PC 系列键盘不是由硬件电路输出按键所对应的 ASCII 码值,而是由单片机扫描程序识别按键的当前位置,然后向键盘接口输出该键的扫描码。按键的识别、键值的确定以及键代码存入键缓冲区等工作全部由软件完成。

目前 PC 机上常用的键盘插口有三种,第一种是比较老式的直径 13 mm 的 PC 键盘插口;第二种是最常用的直径 8 mm 的 PS/2 键盘插口,第三种 USB 接口的键盘,这种键盘现在也逐渐流行起来。

9.1.2 鼠标

鼠标是一种快速定位器,其功能与键盘的光标键相似。通过移动鼠标可以快速定位屏幕上的对象,是计算机图形界面交互的必用外部设备。

1. 鼠标的工作原理

鼠标一般通过微型计算机中的 RS232C 串行口、PS/2 鼠标插口或 USB 接口与主机连接。

鼠标的操作包括两种:一种是平面上的移动,另一种就是按键的按下和释放。当鼠标

器在平面上移动时,通过机械或光学的方法把鼠标器移动的距离和方向转换成脉冲信号传送给计算机,计算机鼠标驱动程序将脉冲个数转换成鼠标器的水平方向和垂直方向的位移量,从而控制显示屏上光标箭头随鼠标的移动而移动。

鼠标驱动程序(Mouse Driver)是鼠标与应用程序之间的接口,属于系统软件,在装入内存后,入口地址存放在中断向量表中,向量码为 33H。在汇编语言程序设计中,可通过软中断指令 INT 33H 调用鼠标驱动程序中的子程序,以实现鼠标的功能。

2. 常见的鼠标类型

鼠标的分类方法很多,常见的有以下几种:

1) 按照按键数目分类

按照按键数目可分为两类:两键鼠标(MS Mouse)和三键鼠标(PC Mouse)。微软认为有两个按键就可以满足使用需要,但 IBM 却认为需要三个按键才够。市场上常见的是二合一鼠标,在鼠标的背面有一开关,开关的两端写着“2”(MS)和“3”(PC),“2”(MS)是指微软的两键鼠标(MS Mouse),“3”(PC)是指 IBM 的三键鼠标(PC Mouse)。可以根据个人的使用习惯,选择两键或三键鼠标,并在两者之间方便地切换。

2) 按接口类型分类

按接口类型可将鼠标分为五类:PS/2 接口、串行接口、USB 接口、红外接口、无线接口。PS/2 鼠标用的是 6 针的小型圆形接口,串口鼠标用的是 9 针的 D 型接口,USB 鼠标使用 USB 接口,具有即插即用特性。红外接口鼠标用红外线与计算机进行数据传输,无线接口鼠标用无线电与计算机进行数据传输,这两种鼠标都没有连接线,故也称为摇控鼠标,使用起来较为灵活,不受连接线的限制。但红外接口鼠标使用时要正对着计算机,角度不能太大,而无线鼠标就没有这个限制。

3) 笔记本计算机用鼠标

笔记本计算机的鼠标分为内置式和外置式两种。其中内置式鼠标又可分为三种:

① 指点杆 它是在键盘中间位置上的一根压感杆,使用者可通过上下左右推动指点杆控制光标的移动。指点杆移动速度快,但控制精度差;

② 触摸屏 采用红外线、电阻、电容等效应将手指的移动方向和距离通知计算机以控制光标的移动。触摸屏没有机械磨损,控制精度一般,但体积较大,必须附着在显示屏上使用;

③ 轨迹球 控制精度高,但易磨损,不易清洁。

外置式鼠标则与普通台式机用鼠标完全相同。

4) 按工作原理分类

按照不同的工作原理,鼠标可以分为以下几种:机械式、光电式和光机式。

(1) 机械式鼠标

在这种鼠标器底部有个被橡胶包盖着的金属球,紧靠着球有两个相互垂直的转轴,在转

轴上装着旋转编码器和相应电路。当鼠标器移动时,球便滚动,使两个转轴旋转,由编码器及相应电路可计算沿水平方向和垂直方向的偏移量。这种鼠标器结构简单、价格便宜、操作方便,但准确度、灵敏度差。

(2) 光电式鼠标

这种鼠标器是通过两对相互垂直的光电监测器中的光敏三极管来检测发光二极管照射到鼠标下面的垫板上产生的反射光。该垫板是画有黑白格子的专用垫板,当发光二极管发出的光线照到黑格子上时,光线被吸收,若光线照到白格子上时,则有反射光,随之光敏三极管依据有无反射光而产生高、低电平,形成脉冲信号。这种鼠标器传送速率快,灵敏度和准确度高,但需用专用垫板,价格贵。

新型的光电鼠标不再使用专用垫板,而是在鼠标底部用一个图形识别芯片时刻监视鼠标与桌面的相对移动,根据移动情况来发出位移信号。

(3) 光机式鼠标

这种鼠标器为光学和机械混合结构,装有滚动橡胶球,不需专用垫板。用两个相互垂直的滚轴紧靠橡胶球上。两个滚轴顶端都装一个边缘开槽的光栅轮。光栅轮的两边分别装着发光二极管和光敏三极管,用于光电检测。当鼠标器移动时,橡胶球滚动,带动滚轴及光栅轮转动。光线通过光栅轮的开槽透光,光线遇到未开槽不透光。从而使光敏三极管产生高低电平,形成脉冲信号。光机鼠标目前是最流行的鼠标种类。

鼠标器最重要的参数是分辨率。它以 dpi(像素/英寸)为单位。表示鼠标器移动 1 英寸所通过的像素数。一般鼠标器的分辨率为 150dpi ~ 200dpi,高的可达 300dpi ~ 400dpi,若屏幕扫描像素为 640×480 时,鼠标器只要移动 1 英寸,则对应屏幕 300 ~ 400 像素位置,基本遍历屏幕的 $2/3$ 。因此鼠标的分辨率越高,鼠标器移动距离就越短。

9.1.3 显示系统

计算机显示系统由显示器和显示适配器两大部分组成,是计算机系统中最重要的输出方式。显示器就是显示图像的部分。显示适配器又称显示接口卡或简称显卡,插接在主机板的扩充插槽上,是主机与显示器之间的接口。

显示器与显卡是两个独立的部件,具有独立的系列标准。

1. 显示器

显示器的作用是将主机输出的电信号经一系列处理后转换成光信号,并最终将文字、图形显示出来。常用的显示器有阴极射线管显视器(CRT)和液晶显示器(LCD)两种。

CRT 显示器又分为荫罩式和电压穿透式。荫罩式 CRT 显示器是最常见的显示器,因其在电子枪与荧光屏间有一个布满栅孔的金属荫罩板而得名。荫罩式 CRT 显示器的特点是

显示分辨率高,价格便宜,使用寿命较长,电源消耗大,体积大。

LCD 显示器采用的技术主要有两种:有源矩阵和无源矩阵。

① 有源矩阵 又称为薄膜晶体管液晶显示器(TFT)。它的每一个像素点都用一个薄膜晶体管来控制液晶的透光率,优点是色彩鲜艳,视角宽,图像质量高,响应速度快。但其成品率低,从而导致价格比较昂贵。

② 无源矩阵 无源矩阵用电阻来代替有源晶体管,制造较为容易。它和有源矩阵相比的最大优势就是价格低。其缺点是色彩饱和度较差,图像不够清晰,对比度也较低,视角较窄,响应速度慢。

LCD 显示器与 CRT 显示器相比较,其特点是外尺寸相同时可视面积更大,体积小(薄),外形美观,图形清晰,不存在刷新频率和画面闪烁的问题,但价格比较昂贵,分辨率较低,响应速度慢。

1) CRT 显示器基本工作原理

荫罩式 CRT 显示器是由三部分组成:阴极射线管、视频放大电路和同步扫描电路。其结构如图 9.4 所示。

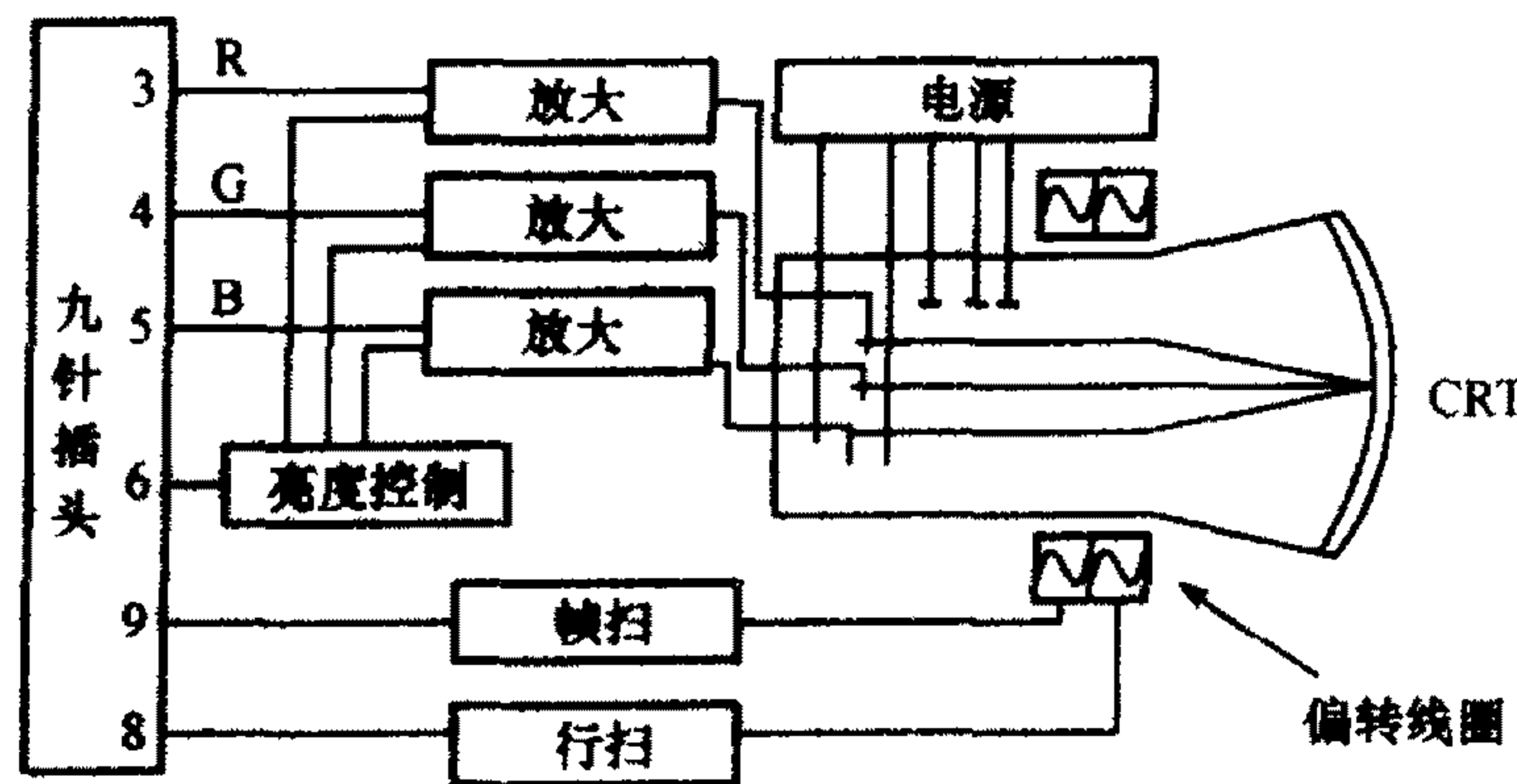


图 9.4 CRT 显示器的结构

阴极射线管由阴极、栅极、加速极、高压极、聚焦极以及荧光屏组成。

当灯丝加热后,由视频信号放大驱动电路输出的电流驱动阴极,使它发射电子束(故又称阴极为电子枪)。由红、绿、蓝三个基色的阴极发射出三色电子束,经栅极、加速极(第一阳极)、高压极(第二阳极)及聚焦极(第三阳极)的作用会聚成很细但能量很大的电子束,高速射向荧光屏,荧光屏上的荧光粉经电子的轰击而发出亮光。荧光屏上的每一个彩色点(即像素)是由红、绿、蓝(R、G、B)三原色组合而成,电子枪发出的三束电子汇聚于荫罩板的小孔或狭缝(即栅孔)中,穿过荫罩板后按不同强度比例点亮荧光粉,从而合成产生各种颜色。大多数显像管的 R、G、B 荧光点都呈“品”字形分布,三原色荧光点连接起来就可以得到一个等边三角形,这就是一个像素点。

行扫描振荡器和帧扫描振荡器在外部信号的同步控制下加到偏转线圈上,使电子束受电磁场的作用从上到下、从左到右有规律地扫描,就形成了光栅。外部信号(R/G/B/亮度)经放大后加到 CRT 的栅极上,控制三个电子束的通过强度,也就控制了不同彩色荧光粉的发光强度,形成丰富多彩的画面。

2) CRT 显示器的指标

(1) 尺寸

现在市场上的 17 英寸、15 英寸的显示器,实际上指的是显像管的尺寸。因为显像管的边框占了一部分面积,实际的有效显示面积到不了标称尺寸。例如,14 英寸的显示器的可视范围只有 12 英寸,17 英寸的显示器的可视范围只有 15 英寸等。

(2) 显像管的形状

根据使用显像管的不同,显示器的屏幕形状也不同,主要有球面、平面直角、柱面和纯平四种。常用的显示器以平面直角和纯平最为常见。

(3) 逐行/隔行扫描

相对于逐行扫描而言,隔行扫描对同一屏幕的图像先扫描奇数行,再扫描偶数行。需要说明的是:隔行扫描器在低分辨率下其实也是逐行扫描的,只有在分辨率提高到一定程度才改为隔行扫描。在高分辨率模式下隔行扫描的图像比逐行扫描的闪烁和抖动得更为厉害,这主要是因为在高分辨率模式下,扫描奇数行时的电子束偏移扫描到偶数行,而扫描偶数行时的电子束偏移扫描到奇数行,因而一条水平线在奇数行和偶数行上下抖动。

(4) 点距

点距是不同像素两个颜色相同的磷光体间的距离。点距越小,显示出来的图像越细腻,分辨率越高。几年以前的显示器的点距多为 0.31 mm 和 0.39 mm,现如今大多数显示器采用的都是 0.28 mm 的点距。高端显示器往往采用更小的点距来提高分辨率和图像质量。如采用 0.25 mm 的点距。

(5) 刷新频率

刷新频率就是每秒屏幕刷新的次数。刷新频率越低,图像闪烁得就越厉害。一般显示器要求在 1024 × 768 像素下要能达到 75 Hz 的刷新率。带宽比较高的显示器其刷新率也较高。

(6) 分辨率

在显示器尺寸一定的情况下,指水平方向和垂直方向上最大像素个数,一般用水平方向像素数 × 垂直方向像素数来表示。

2. 计算机的数据显示过程

计算机屏幕上显示的图像,是光栅扫描过程中将图像分解成按时间分布的视频信号,用来控制电子束在扫描过程中各点的亮度和色彩。为了使图像不消失,显示卡将存放在显示存储器中(VRAM)的一帧视频信息按帧频的速率输出到显示器。显示器也以帧频的速率保

持水平和垂直扫描严格同步,不断地扫描屏幕,便形成稳定的图像。

显示器可实现字符和图形两种显示方式,无论哪种方式,都要求将视频信息存到显示存储器中。

1) 字符显示

屏幕上每个字符都是以光点的点阵形式显示的。每个字符一般由 5×7、8×8、7×9 或更多的点阵图形组成的。图 9.5 中表示的是字符 A 的 8×8 点阵。图中有点处为显示亮点,空白为暗点,这些亮点、暗点分别对应二进制的 1、0 码。所有 ASCII 码字符点阵都存放在显示卡的字模 ROM 中。每个字符点阵占 8 个字节的 ROM 单元。

字符发生器高位地址								字符发生器低位地址			字符发生器内容								
A ₁₀	A ₉	A ₈	A ₇	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀ (点阵行地址)		7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	0	0	0	10H				●				
0	1	0	0	0	0	1	0	0	0	1	28H			●		●			
0	1	0	0	0	0	1	0	0	1	0	44H		●				●		
0	1	0	0	0	0	1	0	0	1	1	82H	●							●
0	1	0	0	0	0	1	0	1	0	0	FEH	●	●	●	●	●	●	●	
0	1	0	0	0	0	1	0	1	0	1	82H	●							●
0	1	0	0	0	0	1	0	1	1	0	82H	●							●
0	1	0	0	0	0	1	0	1	1	1	00H								

图 9.5 字符 A 的 8×8 点阵

在字符显示方式下,显示存储器 VRAM 中存放一帧或几帧待显示的 ASCII 码字符。VRAM 中的字符顺序与屏幕显示的位置是一一对应的,如图 9.6 所示。依据字符在屏幕上显示的行、列位置,决定它在 VRAM 中的顺序位置。在 VRAM 中的两个字节对应屏幕上的一个显示字符(见图 9.5)。高字节作为字符属性,确定字符和字符底色的颜色;低字节为字符的 ASCII 码,送到字符发生器作为高位地址 A₁₀~A₃,字符发生器的低位地址由字符点阵行地址决定。

2) 图形显示

在图形显示方式下,VRAM 中存放的是要显示的图像数据,数据的顺序与屏幕上图像的行列位置一一对应。屏幕上的图像被划分为几百个水平的和几百个垂直的像素点,每个像素的位置是由该点的行、列地址来确定的。当像素点位置确定后,其在 VRAM 中的位置也随之确定了。

假设每一屏由 640×200 个像素组成(每行 640 个像素,共 200 行),则每个像素的位置(行、列值)是在(0,0)~(199,639)范围内。如果每个像素只能是黑、白(单色)显示,那么屏

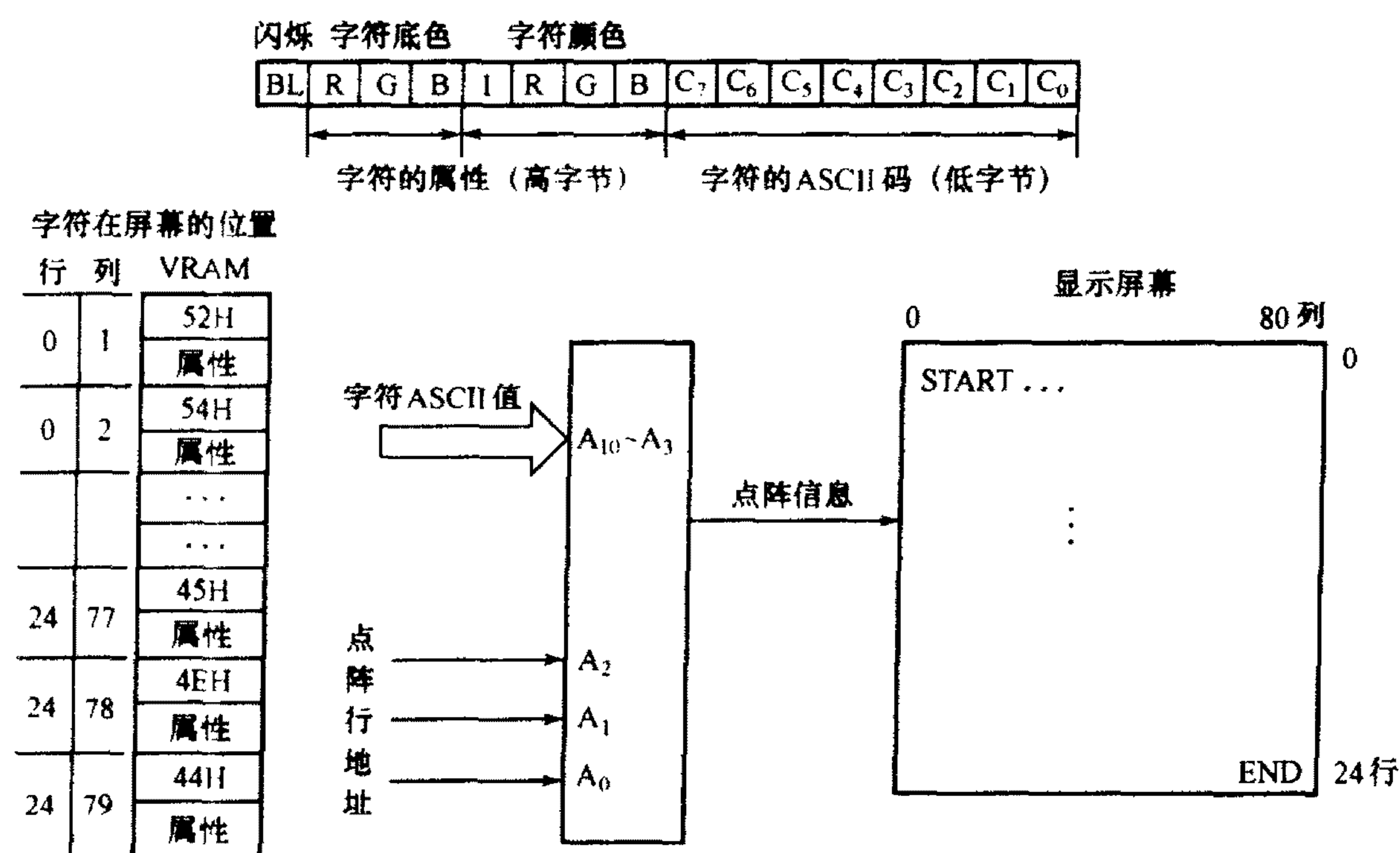


图 9.6 显示屏与显示缓存的关系

幕上的一个像素可用 VRAM 中的一个比特来表示,1 表示亮点,0 表示暗点。屏幕上一个扫描行占据 $640/8 = 80\text{B}$,整屏占据 $80 \times 200 = 16\,000\text{B}$ 。由于 VRAM 中的内容与屏幕上的亮暗点一一对应,所以显示时直接取出 VRAM 中的内容转换成视频信号输出即可。如果是彩色图形显示方式,那么屏幕上每个像素就要用 VRAM 中的若干个二进制位来表示,位数越多,能显示的色彩数就越多。例如,用 4 个比特表示一个像素时,可显示 16 种色彩。用 16 个比特表示一个像素时,则可显示 $2^{16} = 65\,536$ 种色彩。显然,色彩数越多,需要的显存容量就越大。

当屏幕的垂直点数 $M \times$ 水平点数 N 和彩色数 C 确定后,所需的 VRAM 容量就可以通过下式计算出来:

$$\text{VRAM 容量} = (M \times N \times \log_2 C) / 8(\text{B})$$

3. 显示适配器

1) 显示适配器的构造

显示适配器又称显示卡,是显示器与主机的接口,为显示器提供各种信号。显示卡通过插座和主机的系统总线相连,同时在卡的背面又通过 9 针 D 型插座与显示器连接。

显示接口卡上包括 CRT 控制器(CRTC)、定时器、ROM 字模、VRAM、视频 BIOS 等部件。CRTC 是一个高度集成的显示控制器,它产生控制信号并管理显示卡的操作。显示内存 VRAM 用来存储将要显示的文字或图像数据。ROM 字模用来存放文本字符的点阵格式。当显卡工作在图形模式下时,ROM 字模无效。视频 BIOS 用来存储进行视频操作的指令和

程序。

在计算机加电自检期间,系统完成了对 CRT 控制器的初始化。在建立了显示方式、进行相应 VRAM 自检后,显示接口卡在 CRT 控制器控制下,按照编程设置的工作方式独立控制显示器工作,为显示器提供所需的视频信号和同步信号。

早期的 CRT 控制器芯片常采用 MC6845 CRTC。新一代的显示控制器则以 Nvidia 公司的 GeForce 为典型代表。它不仅能提供强大的二维图形显示能力,而且提供了强大的三维显示能力。

2) 显卡的技术指标

(1) 图形加速能力

传统的显示卡过多地依靠系统 CPU 进行图形处理,因此限制了整个计算机系统的图形处理能力。新型的显卡往往采用图形加速芯片来代替 CPU 完成画线条、椭圆、多边形、区域填充等图形处理功能。

图形加速芯片直接与计算机扩展总线相连,图形命令和数据被图形加速芯片转换为像素数据存储在显示内存中,然后由数模转换器(DAC)把显示内存中的像素数据转换为红、绿、蓝三色模拟信号与合成的水平及垂直同步信号一起驱动显示器。

以画线为例,若没有图形加速芯片,主板上的 CPU 就需要计算出线条上每一点的位置和颜色数据,然后再通过主板上的总线将这些图形数据传输给显示卡。若显示卡上有图形加速功能,则主板上的 CPU 只需将线条的两端点位置、颜色和画线命令传输给显示卡,然后由图形加速芯片硬件完成画线工作,速度比软件快得多。

现在常见的图形加速芯片有 Voodoo 系列、GeForce 系列、Radeon 系列、MGA 系列等。

(2) 接口总线类型

图形数据必须在内存和显示卡之间传输。数据传输的通道就是主板的扩展总线。如果总线的数据传输速率高,视频性能就会大大提高。因此,总线的结构对视频性能产生了很大的影响。显示卡的总线接口类型按出现的顺序分别有 ISA、VESA、PCI 和 AGP。ISA、VESA 现在已被淘汰,PCI 总线接口的显示卡是目前的主流产品,可满足大多数应用的需要,但对处理大量图形的工作,如 CAD、动画制作等,最好选用 AGP 接口的图形显示卡。

(3) 显存的数量及类型

显示卡上的显示内存用来存储显示屏上各点的图形和颜色信息,因此,要显示的分辨率越高,颜色越多,显示内存也就越大。例如,每屏扫描像素为 1024×768 ,16 位彩色所需的显存为 $1024 \times 768 \times 16/8 = 1\,573\,864\text{ B} \approx 1.57\text{ MB}$,比较实用的显示像素为 640×480 、 800×600 ,颜色深度为 256 色,使用 1 MB 显示内存已可以。但为了显示图形和游戏动画的需要,现在的 3D 显示卡上大多装备了 8 MB ~ 64 MB 显存以满足高质量图形的输出。

显示卡依靠显示内存来保存图像数据,较快的显存允许对图像数据较快地读/写,从而

提高显示卡的性能。显示卡上常用的显存类型按速度快慢分别有 SDRAM (Synchronous DRAM)、DDR DRAM (Double Data Rate DRAM)、RDRAM (Rambus RAM) 等。较高档次的显卡多采用 DDR RAM 或 RDRAM 作为显存。

9.1.4 打印机

打印机也是计算机系统中标准的输出设备之一, 可用来打印字符、数字、图形和表格等。打印机种类很多, 按照打印原理, 可分为击打式打印机和非击打式打印机。

击打式打印机是用机械方法, 使打印针或字符锤击打色带, 在打印纸上印出字符。非击打式打印机是通过激光、喷墨、热升华、热敏等方式将字符印在打印纸上。

1. 打印方式

打印机与主机之间的数据传送, 既可以是并行传送, 也可以是串行传送。目前, 大多数的打印机是并行数据传送的, 它们通过并行接口与主机连接。而对部分串行打印机, 则通过串行口连接到主机。

同显示器一样, 打印机在微型计算机系统的工作方式也可按其从主机接受的数据类型分为字符方式和图形方式。

所谓字符方式, 是指主机在发送打印数据时, 只传送字符的 ASCII 码, 打印机根据收到的 ASCII 码从字模 ROM 中取出相应的字符点阵信息, 最后用机械、光学或加热的方法打印到纸上。汉字的打印也可以在字符方式下进行, 这要以打印机内部具备全部汉字字模为前提。字符方式可以获得较快的打印速度, 是当前西文打印中最常用的方法, 中文打印如采用这种方式, 打印机的成本就要相应提高。字符方式不能用于图形打印。

在图形方式下, 主机所传送的不是字符代码, 而是经过软件编辑的图形像素信息。图形方式既可以打印西文字符, 也可以打印汉字或任意的图像。

在微型计算机系统中, 上述两种打印方式往往是共存的, 到底使用哪一种方式要视具体情况而定。有时, 用户可用键盘输入命令或通过程序中给定的指令来选择其一, 有时由系统规定而不能改变。

2. 并行打印机接口

打印机通过接口与主机相连, 该接口也称为打印机控制器或适配器。它可以是一块独立的选件板, 也可以是包括显示控制器、打印机控制器、软盘控制器及硬盘控制器等多功能卡的一个组成部分。它插接在主机板的扩充槽内, 一端与系统总线相连, 另一端则通过外部总线接到打印机上, 其连接采用标准的 25 芯插头插座。

适配器与打印机之间的接口信号可分为 3 组, 即:

- ① 8 位数据信号 用于传送数据字符和控制字符;

- ② 4 位状态信号 用于传送 CPU 到打印机的各种控制信号；
 - ③ 5 位控制信号 负责传送打印机向 CPU 发送的当前状态的信号。
- 另外的 8 位是接地信号。

在 CPU 与打印机进行数据传送时,首先要由接口向打印机提供“选择输入”控制信号,打印机在此信号控制下,才能接收数据及其他控制信号;同时,打印机要向接口送上有效的“打印机选中”状态信号,表示打印机已加电工作。

在此之后,CPU 可通过接口向打印机输出字符。在每一次数据选通信号 $\overline{\text{STROBE}}$ 有效期间,CPU 向打印机内部的缓冲存储器输出一个字符的 ASCII 码,打印机接收到之后向 CPU 发送应答信号 $\overline{\text{ACK}}$ 。打印机在接收一行字符期间及在收到回车符之前,无机械动作,忙信号 BUSY 为低电平有效;在收到回车符之后,打印机在连续打印一串字符及回车、换行等机械动作期间,BUSY 为高电平,此时打印机不能再接收数据,直到打印机发出应答信号后,又可接收下一批字符(也有的打印机是在 $\overline{\text{STROBE}}$ 下降沿启动 BUSY 信号变高电平)。图 9.7 给出了遵守 Centronics 标准的打印机时序。

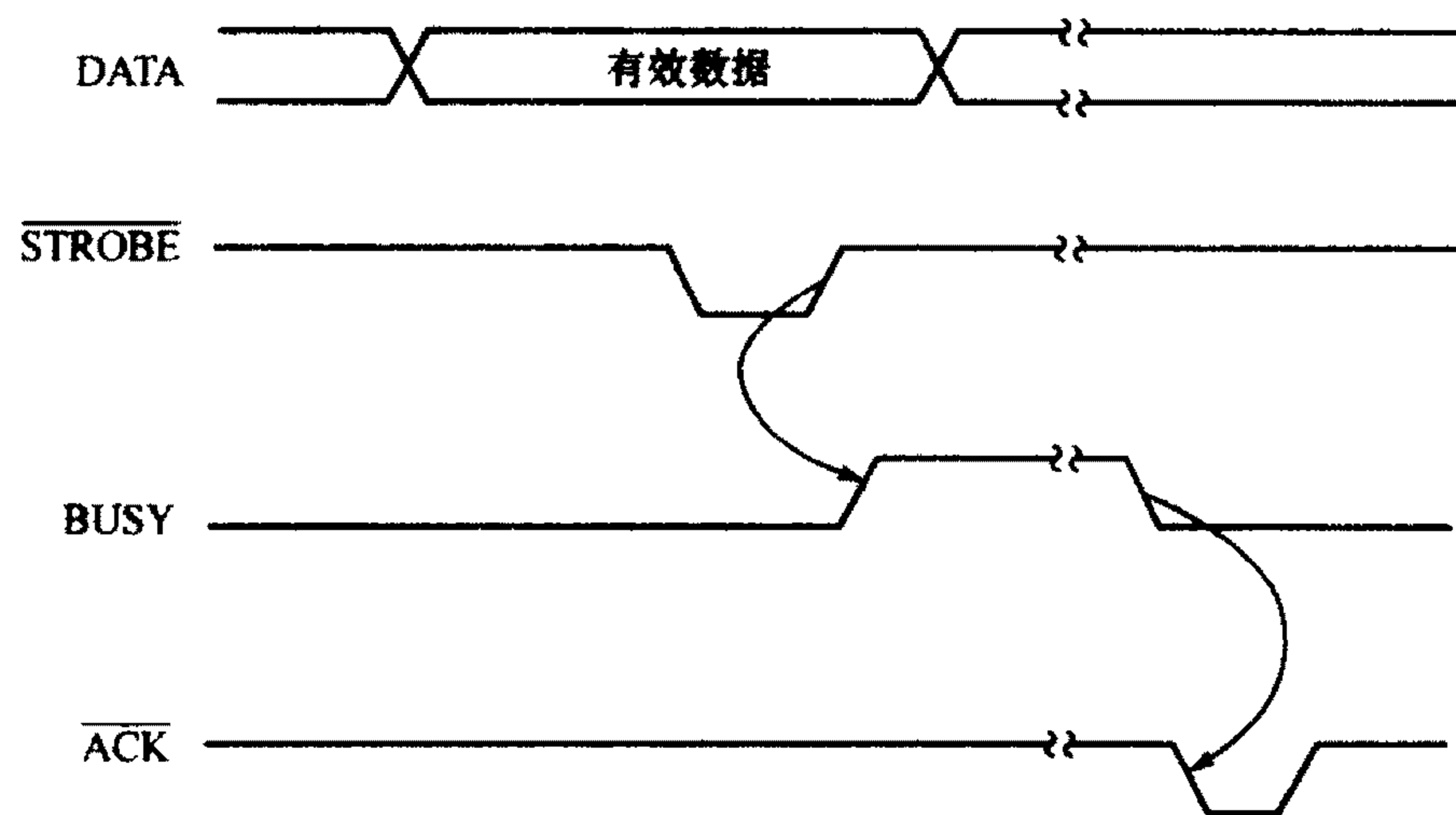


图 9.7 Centronics 标准的打印机时序

3. 打印机的主要性能指标

一般而言,衡量打印机性能的主要指标包括以下几个方面。

1) 分辨率

分辨率用 DPI 表示,即每英寸打印点数,它是衡量打印质量的重要指标。不同类型的打印机其打印质量也不同。针式打印机的分辨率较低,一般为 180dpi ~ 360dpi。喷墨打印机分辨率几乎是针式打印机的两倍,一般可达 300dpi ~ 720dpi。激光打印机的分辨率为 300dpi ~ 2 400dpi。

2) 打印速度

针式打印机的速度用每秒打印字符数 cps 表示。打印速度在不同的字体和文种下差别较大。针式打印机的打印速度由于受机械运动的影响,在印刷体方式下一般不超过 100cps,在草稿方式下可以达到 200cps。

喷墨打印机和激光打印机都属于页式打印机(即计算机输出完一整页的内容,打印机才开始打印),打印速度以每分钟打印页数(ppm)表示,一般在几 ppm 到几十 ppm 之间。

3) 汉字打印、中西文字库及打印字体

能否打印汉字是衡量打印机性能的一项重要指标。有无中文字库对打印机的打印速度影响很大。另外,打印字体也是一个影响速度的因素。目前针式打印机打印汉字字体最少有四种(宋体、仿宋体、楷体、黑体),打印各类英文、数字字符有 5~10 种;喷墨打印机打印的西文字体有 6~8 种,中文 3 种以上;激光打印机有 3 种中文字体(宋、楷、黑)及各种英文字体。以上均指打印机自带字库的情况,若使用图形打印方式,则打印字体仅与主机支持的字体数量有关。

4) 打印缓冲存储器

打印机设置较大的缓冲存储器是为了满足高速打印和打印大型文件的需要。缓冲存储器的大小将影响打印速度。针式打印机的缓冲存储器一般为 16 KB。喷墨打印机和激光打印机的缓冲存储器因在图形方式下要存储大量的图形点阵信息,并且是整页装入,其缓冲存储器较大,通常容量可达 4 MB~16 MB。

5) 打印幅面

打印幅面问题是用户直接关心的问题。对针式打印机,规格有两种:80 列和 132 列,即每行可打印 80 个或 132 个字符。对非击打式打印机,幅面一般为 A4、A3 和 B4。

6) 接口类型

打印机的接口类型主要有三种:并行接口、串行接口和 USB 接口。并行接口应用最广泛,所以人们往往把计算机上的并行接口俗称为打印机接口。

4. 几种常见打印机

目前市场上常见的打印机有点阵式、喷墨和激光打印机三种。点阵式打印机现在主要用于银行、税务等部门的票据类打印,喷墨和激光打印机则因其打印性能、效果等方面的优势而越来越得到广泛的应用。

1) 点阵式打印机

点阵式打印机也称针式打印机,在结构上主要由带动打印头的步进电机、走纸步进电机、色带及接口控制电路组成。打印机原理如图 9.8 所示。

点阵式打印机的打印头是一列打印针组成的,打印头有 9 针、16 针、24 针等几种,打印针排成一个垂直列。打印时,打印头横向移动,打印针每次一列地按纵向打印字符点阵,当

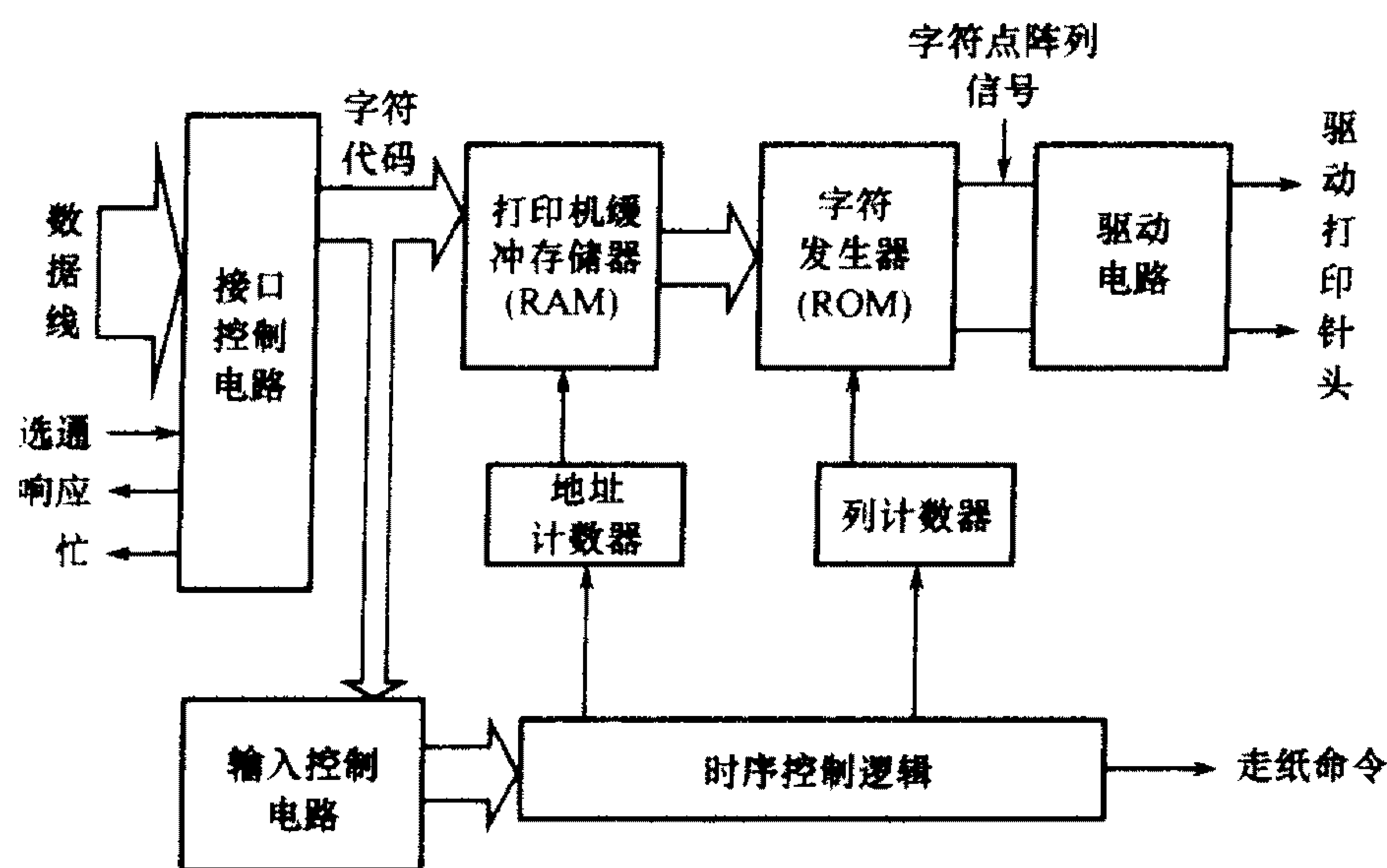


图 9.8 点阵打印机控制逻辑框图

一行字符点阵打印完毕,走纸到下一行,打印头回到行首准备打印下一行。

点阵打印机分为字符和图形两种打印机。其中 16 针、24 针一般都属于可打印图形的打印机。具有图形打印功能的打印机才能打印汉字。

接口控制电路的功能是接收计算机送来的打印数据并向计算机报告打印机状态。

计算机送来的数据包括打印字符和控制字符两种,打印字符直接送至打印机缓冲存储器,控制字符则送至输入控制电路,用于产生相应的打印控制信号。

与打印机的一般工作过程类似,点阵式打印机的字符打印过程为:

主机检查打印机状态,若打印机处于“忙”,则主机等待;若处于“不忙”,则主机输出数据并同时发出“选通信号”。选通信号用于将主机输出的数据送入打印机缓冲存储器,并在此之后向主机返回一个“应答信号”,通知主机可发送下一个数据。如此重复传送数据,直到打印机缓冲存储器满时,接口电路回答“忙”信号,则主机停止发送数据,打印机进入打印阶段。

打印机开始打印时,先在缓冲存储器中取一个 ASCII 码,作为字符发生器的高位地址,列计数器作为字符发生器的低位地址,从字符发生器取出字符的一列点阵信息,送至驱动电路,驱动打印针头打印出该字符一列的点。每打印一列,列计数器加 1,字符发生器依据列计数器的值,依次取出字符点阵各列信息。打印完一个字符后,地址计数器加 1,再取出下一个字符打印。打印头在打印时序电路控制下,自左向右边打边移动,一行打印完后,发出走纸命令,然后打印头返回到最左端,这样开始重复输入新一行数据。

图形打印时,主机发送的打印数据本身已经是点阵数据,因此可将缓冲存储器中的点阵数据直接送到驱动器,控制打印头的动作。

2) 喷墨打印机

喷墨打印机使用很细的喷嘴,用加热的方法使墨水沸腾喷在纸上完成印字。墨水喷嘴排成一个纵列点阵。打印时,根据要打印的内容使点阵中要印出墨点位置的墨水微粒不带电,而不印墨点的位置的墨水微粒带电。这样,当墨水微粒经过电场时,带电的微粒被吸附下来;未带电的微粒按点阵字的形式凝固在纸上形成字符。彩色打印时,黑、红、黄、绿色墨水一起喷点,则可打印出彩色图形。喷墨打印机的优点是字迹清晰、美观、速度快,打印机价格较低。

3) 激光打印机

激光打印机使用激光技术和电子照相技术实现打印,原理类似于复印机。是一种高精度、高速度、低噪声的非击打式打印机。激光印字机的工作原理如图9.9所示。它主要由激光扫描系统、电子照相系统和控制系统三部分组成。

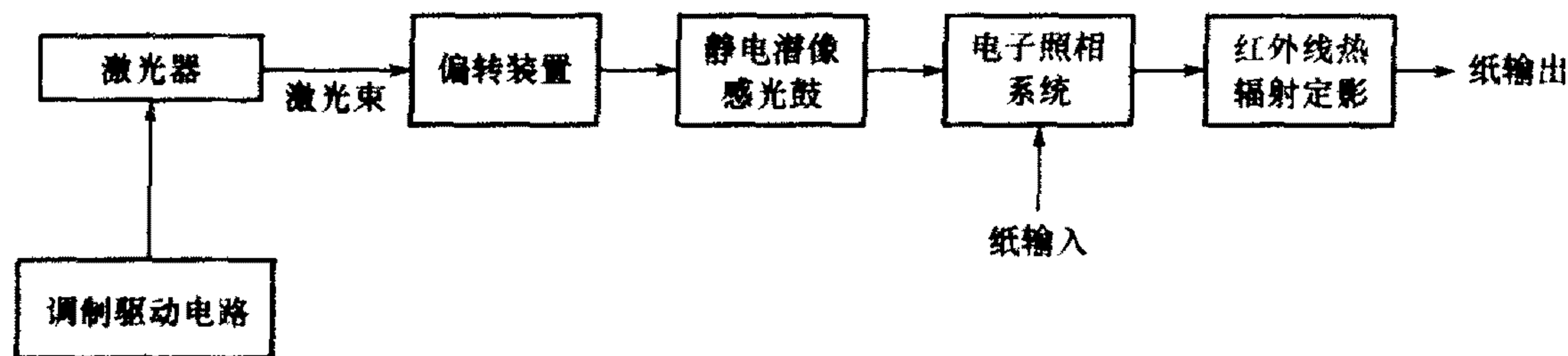


图 9.9 激光打印机的工作原理图

激光扫描系统主要作用是使激光器产生的激光经调制后,变成载有字符或图形信息的激光束,该激光束经扫描偏转装置在感光鼓上扫描,形成“静电潜像”。

电子照相系统使带有“静电潜像”的感光鼓接触带有相同极性电荷的干墨粉,鼓面原被激光照射的部位将吸附墨粉,便显影出图像。该图像转印到纸上,经红外线热辐射定影后,使墨粉渗透到纸纤维中固定。

控制系统包括激光扫描控制、电子照相系统控制、缓冲存储器和接口控制等。控制系统完成接收和处理主机的各种命令和数据,以及向主机报告打印机状态。

9.1.5 网卡

网络接口卡(Network Interface Card, NIC)简称网卡,是插在计算机总线插槽内或某个外部接口上的扩展卡,它与网络程序(网络操作系统)配合工作,控制网络上信息的发送与接收。网卡上一般有一个或多个网络接口,用来把计算机连接到网络上。

1. 网络的基本概念

计算机网络是以资源共享和信息交换为目的、通过数据通信线路将多台计算机互连而成的系统。计算机网络中可以共享的资源包括三类：硬件资源、软件资源和数据资源。其中，共享硬件资源是共享其他资源的物质基础。

共享的硬件资源包括：大容量存储器、各种外部设备以及主机的运算能力等；软件资源包括：各种语言处理程序、服务程序和应用程序；数据资源包括：各种数据文件和数据库等。

主要的计算机网络类型有两种：

① 广域网(Wide Area Network, WAN), 覆盖范围一般为几十到几千 km。

② 局域网(Local Area Network, LAN), 覆盖范围通常为几 m 到几十 km。

而现在发展迅速的因特网(Internet)则是由千千万万个局域网和成千上万台主机系统互连而成的遍及全球的巨型网络。它所使用的网络通信协议(Protocol)称为 TCP/IP 协议。

局域网是一种共享介质类型的网络, 即局域网上的所有计算机共享同一个传输信道。为了使所有计算机能够有秩序地发送数据, 避免互相冲突, 需要有一种规则来协调网络上每一台计算机对共享信道的访问, 这种规则就称为介质访问控制方法。

根据不同的介质访问控制方法, 局域网分为以太网(Ethernet)、令牌环网(Token Ring)、光纤分布式接口(FDDI)和异步传输方式(ATM)等。其中以太网是目前国内应用最广泛的一种局域网。它采用的介质访问控制方法称为“带冲突检测的载波侦听多路访问规程(CSMA/CD)”。

将一台计算机接入网络, 需要的设备与所连接的网络类型有关。例如, 要将一台计算机接入广域网, 一般需要使用调制解调器; 如果是接入局域网则要利用网络接口卡, 而要把一台计算机与以太网相连则要使用以太网接口卡。本节主要介绍以太网卡。

2. 以太网卡的结构

以太网卡主要包括以下几个部分：

- ① 发送和接收部件；
- ② 载波检测部件；
- ③ 发送和接收控制部件；
- ④ 曼彻斯特编码/译码器；
- ⑤ LAN 管理部件；
- ⑥ 微处理器(有些网卡无此部件)。

这些部件互相配合, 实现了以太网的 CSMA/CD 介质访问控制方法。以太网卡的结构如图 9.10 所示。

每个网卡在出厂时都被赋予了一个全世界范围内惟一的地址, 称为网卡的网络地址(也

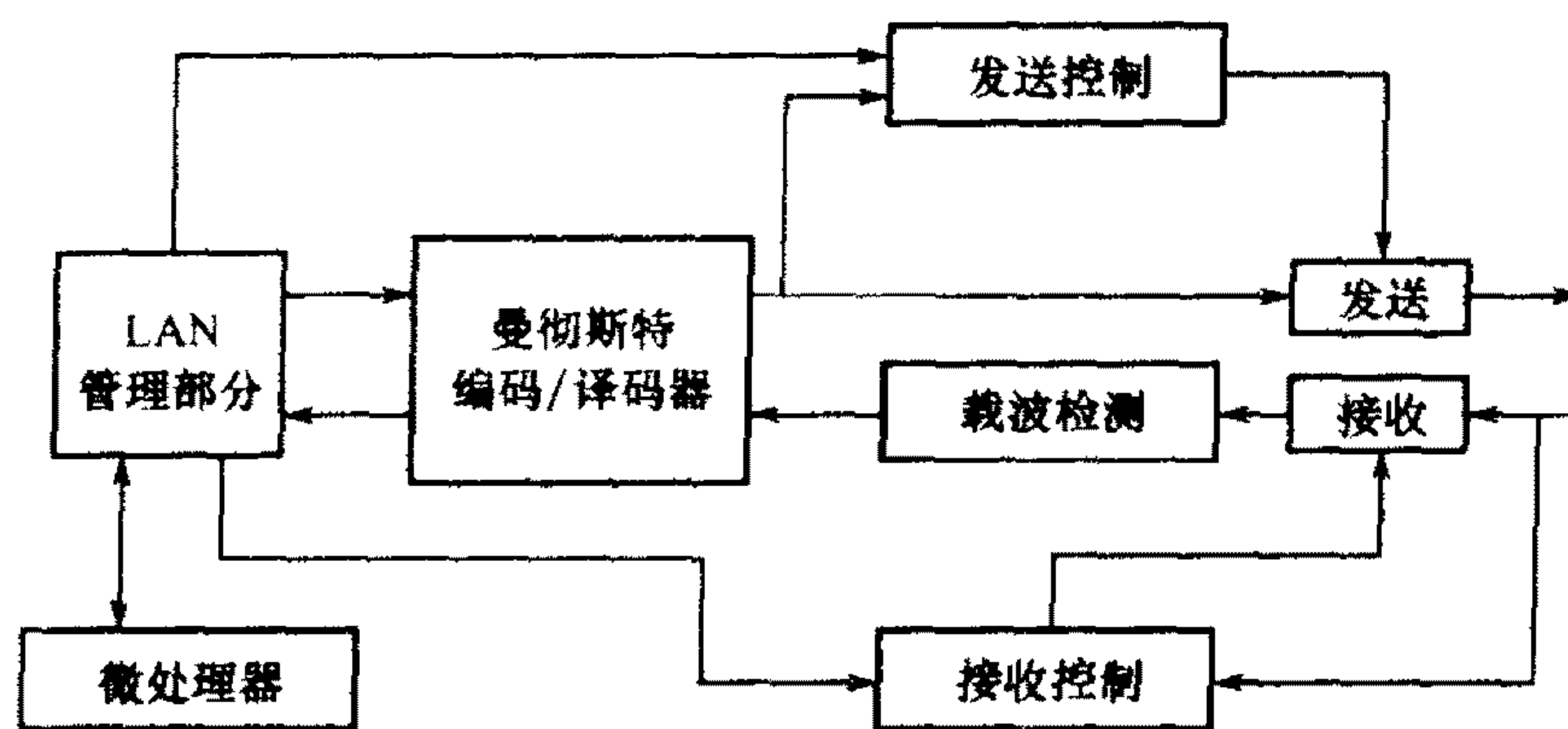


图 9.10 以太网卡结构

称为 MAC 地址)。所有网卡制造商对网卡地址范围达成协议,每个制造商只能使用许可范围内的地址,这样可保证生产出来的网卡不使用重复的地址。网卡地址是一个 48 位的二进制数,被固化在网卡硬件中。

网卡既是一种接口,它在系统中就占有一定的端口地址。在网卡上通常都提供有多个 I/O 端口地址,以供选择使用。常用的缺省 I/O 地址是 300H。

当工作站需要远程引导启动时,需要在网卡上插一片远程引导 ROM 芯片,这个 ROM 芯片必须映射到计算机 640 KB~1 MB 之间的存储区上。为了实现映射,一是要允许网卡进行远程引导,二是要规定存储空间的基地址。这个基地址就是网卡的 ROM 芯片要映射到的存储区的起点。

3. 网卡的选用

1) 总线类型

以太网卡按总线宽度可分为 8 位、16 位和 32 位网卡。目前 8 位网卡已经被淘汰。工作站上通常采用 16 位或 32 位网卡,服务器则以采用 32 位网卡为主。

按总线类型又可分为 ISA、EISA、MCA、PCI、PCMCIA 和并行接口网卡。由于总线类型不同,在选择网卡时需要与使用的计算机相匹配。目前常见的网卡以 PCI 总线的居多,PCI 总线网卡不仅具有明显的性能优势,而且支持“即插即用”。

在选用网卡时,建议在服务器上使用 PCI 或 EISA 总线的智能型网卡,工作站上则可考虑 PCI 或 ISA 总线的普通网卡,若不考虑传输速度,只是为了安装方便,也可选用并行接口的便携式网卡。对于笔记本电脑,则以 PCMCIA 总线的网卡为首选,然后是并行接口的便携式网卡。

2) 接口类型

由于网络的传输媒介不同(常见的有粗同轴电缆、细同轴电缆、无屏蔽双绞线三种类

型),一般的网卡都有三种接口类型供选择(有些网卡在一块卡上同时提供两种端口,甚至有的在卡上同时提供三种接口)。这些接口分别是:AUI接口,用于连接粗同轴电缆;BNC接口,用于连接细同轴电缆;RJ-45接口,用于连接双绞线。

以前许多小型局域网都采用细同轴电缆和BNC接口的总线网,它成本较低,但是故障率高(一点出错就可能導致整个网络瘫痪),不便于维护和管理。

目前的办公室环境(尤其是像智能大厦这样的综合布线系统)几乎都采用无屏蔽双绞线作为传输介质,接口为RJ-45,网络拓扑结构为星形,这种采用双绞线的星形结构网络目前使用极为广泛。其优点是可靠性高,安装密度高,使用方便,便于集中管理。

对于采用光纤传输介质的网卡,常见的接口类型为SC或ST型接口。

3) 网络速率

对于以太网卡,常见的传输速率等级有10 Mb/s、100 Mb/s和1 000 Mb/s三种。还有一种10/100 Mb/s自适应网卡,它可根据网络的速率自动工作在10 Mb/s或100 Mb/s速率上。

以太网卡的分类情况如图9.11所示。

4. 网卡的安装和配置

网卡具有一组配置选项以保证网卡能与计算机中的其他部件协同工作。这些选项主要包括IRQ(中断请求)、I/O地址、存储器基地址和DMA通道号。

IRQ是这些参数中最重要的一個参数。其缺省配置一般为IRQ₃,在大多数情况下这个值无需改变。若系统中有其他部件也使用了IRQ₃,就需要将网卡的IRQ值改变为其他未被使用的值。

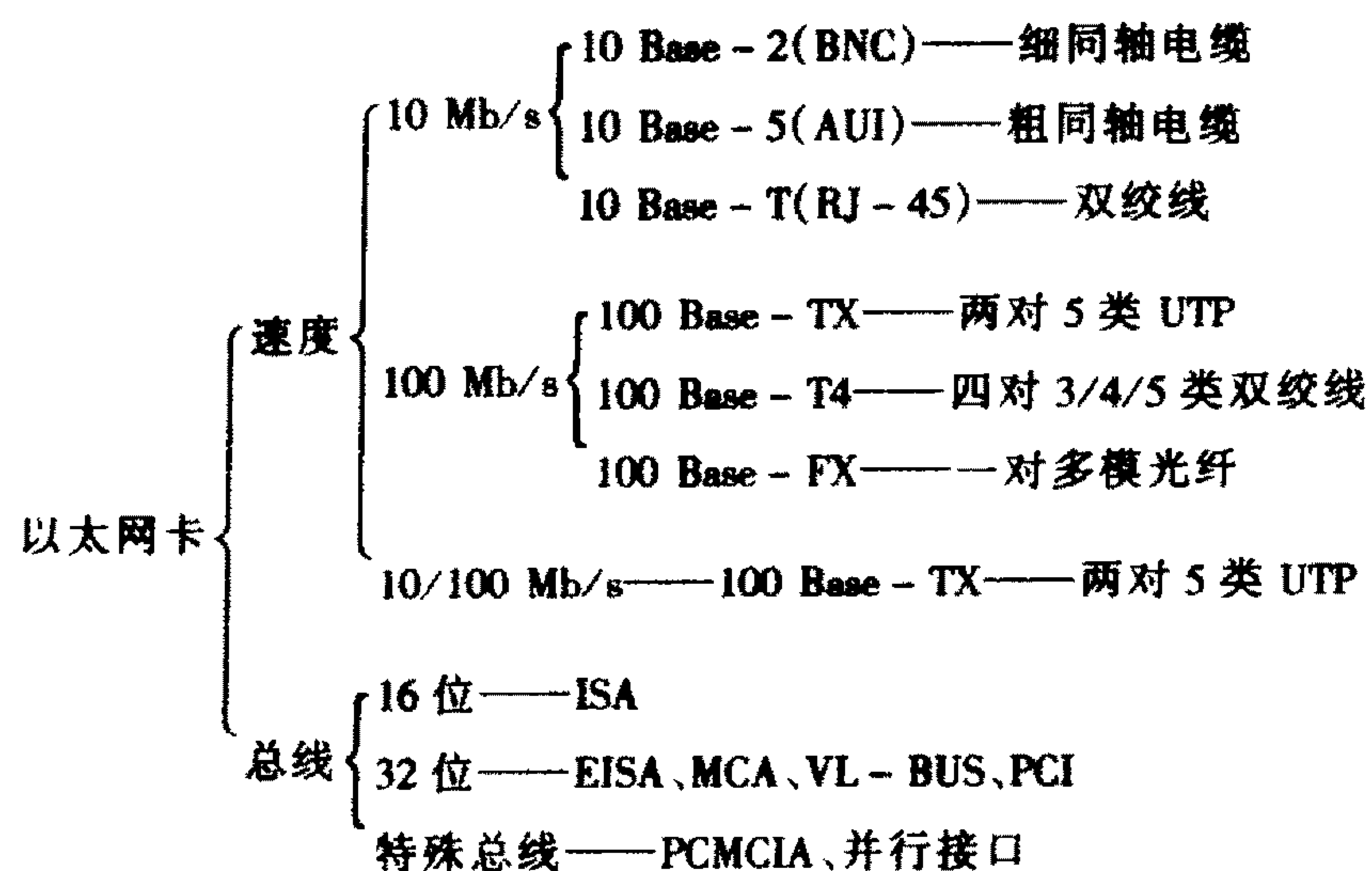


图 9.11 以太网卡的分类

I/O 端口地址用来访问网卡上包含的状态寄存器和控制寄存器,以便使工作站了解网

卡的工作状态和对网卡实施控制。在工作站上除具有网卡外还有其他外设,因此,在选择 I/O 端口地址时,要避免冲突。

存储器基地址用于将远程引导 ROM 映射到计算机的内存空间。若是一般的计算机,这个值不用设置。DMA 通道号是仅在网卡采用 DMA 方式与内存交换数据时使用,一般由系统自动指定。

9.1.6 调制解调器

1. 概述

对与家庭上网,大多利用现有庞大而成熟的公用电话网。目前的电话入户信号都是模拟信号,而计算机所处理的信息是数字信号,因此家庭用户的计算机在联网时,必须有能将数字信号转换为模拟信号及将模拟信号转换为数字信号的转换装置,前者叫调制,后者叫解调,把两种功能做在同一设备上,这种设备就叫调制解调器,即 MODEM。在电话拨号线路上传输数字信号,MODEM 是不可缺少的设备。

传统的电话线调制解调器作为连接计算机和因特网的主要设备,以其使用方便、价格适中、功能齐全等特点,对因特网的普及和发展起到了极其重要的作用。随着网络应用的不断深入,调制解调器在传输速率上也有了极大的提高,性能上日趋完善。在短短的二三年间,调制解调器的速度就从 2.4 kb/s 发展到了现在的 56 kb/s。

2. 调制解调器的工作原理和标准

MODEM 与大多数通信系统一样,采用正弦信号作为载波,将数字信号调制到载波上进行传送。调制方式包括三种基本调制方式:调幅(AM)、调频(FM)和调相(PM),以及由此派生出的其他调制方式。

调制,有模拟调制和数字调制之分。模拟调制对载波信号的参量进行连续地估值;而数字调制使用载波信号的某些离散状态来表征所传送的信息,在接收端对载波信号的离散调制参量进行检测。数字调制信号称为键控信号,由此而形成振幅键控(ASK)、频移键控(FSK)和相移键控(PSK)三种基本方式。

为了便于调制解调器发展的标准化和规范化,国际电信联盟(ITU,原为 CCITT)经过多年努力,逐步完善了有关调制解调器的 V 系列建议,对调制解调器的相关系列做了详尽的规定。目前世界上所有的调制解调器生产厂商,均遵照 V 系列建议中的规定生产。

符合 ITU V 系列建议的调制解调器,大体上采用了以下几种调制方式:

1) 频移键控(FSK)调制

频移键控调制方式主要用于低速、异步传输的调制解调器,相应的 ITU V 系列建议主要有 V.21 和 V.23 两种,分别对应于调制速率为 300 b/s 和 600 b/s,1 200 b/s 的调制解调器。

2) 差分移相键控调制(DPSK)

差分移相键控调制(DPSK)多用于中速调制解调器,主要的类型有 V.22、V.26 和 V.27 建议系列。其速率在 1 200 b/s ~ 4 800 b/s 之间。

3) 调相调幅相结合的调制(PSK)

调相调幅相结合的调制(PSK)一般用于中速的调制解调器。ITU - V 建议是 V.29。传输速率为 9 600 b/s,可降速至 7 200 b/s 或 4 800 b/s 操作。

4) 智能调制解调器

在 20 世纪 80 年代之后,网络编码调制理论有了巨大的进展,并且电子元器件在集成化方面也取得了突破,加上自适应均衡技术和压缩技术的改进和提高,为智能调制解调器的问世创造了条件。ITU 也同时为智能调制解调器的标准化和规范化提出了一系列相应的建议,如: V.32、V.32bis、V.24、V.42、V.42bis 以及最新制定的 V.90 等,它们的传输速率在 14.4 kb/s 到 56 kb/s 之间。

3. 高速调制解调器技术

调制解调器的传输速度最初只有 300 b/s。随着技术的发展,1996 年出现了 33.6 kb/s 的调制解调器产品。此时的 33.6 kb/s 速率认为是调制解调器的极限速率,不会再有突破。而就在 1996 年底,56 kb/s 调制解调技术问世了。新的 56 kb/s 调制解调器的出现,计算机行业中引起了极大反响,成为人们关注的焦点。为了推进 56 kb/s 调制解调器的发展,ITU 于 1998 年颁布了 V.90 建议,正式建立了 56 kb/s 调制解调器的标准。下面简单介绍 56 kb/s 调制解调器技术的原理。

早在 1948 年,香农(Shannon)就用信息论的理论推导出了带宽受限且具有高斯白噪声干扰的信道的极限信息传输速率。由香农理论可知,无论采用何种先进的编码技术,模拟电话线路的信息传输速率一定不可能超过 35 kb/s 的极限。这就是为什么人们认为 33.6 kb/s 速率认为是调制解调器的极限速率,不可能再有突破的原因。那么现在的 56 kb/s 调制解调器技术到底是如何突破理论极限传输速率的呢?

早期的电话交换网采用的是模拟传输。随着技术的进步,电话公司已逐步用数字电路取代原始的模拟网络。但用户住所到市话局之间的这段线路要实现数字化,还需相当长的时间,因而无法实现全数字的通信。就目前来说,电话网络惟一的模拟部分是将用户与市话局连接起来的电话线,如图 9.12 所示。各种调制解调器标准(包括最高速率为 33.6 kb/s 的 V.34 bis 标准)就是运行在这种由两个模拟用户与数字网络相连的交换式网络模型上的。用户的模拟信号通过模数转换器(ADC)转换成数字形式,电话交换网络使用 64 kbit/s 数字通道在用户之间传送数字化的模拟信号。对于这种网络模型来说,33.6 kb/s 被认为是最大的可用传输速率。这种局限是由于用户环路中电话公司一端的 ADC(模数转换器)所造成的。

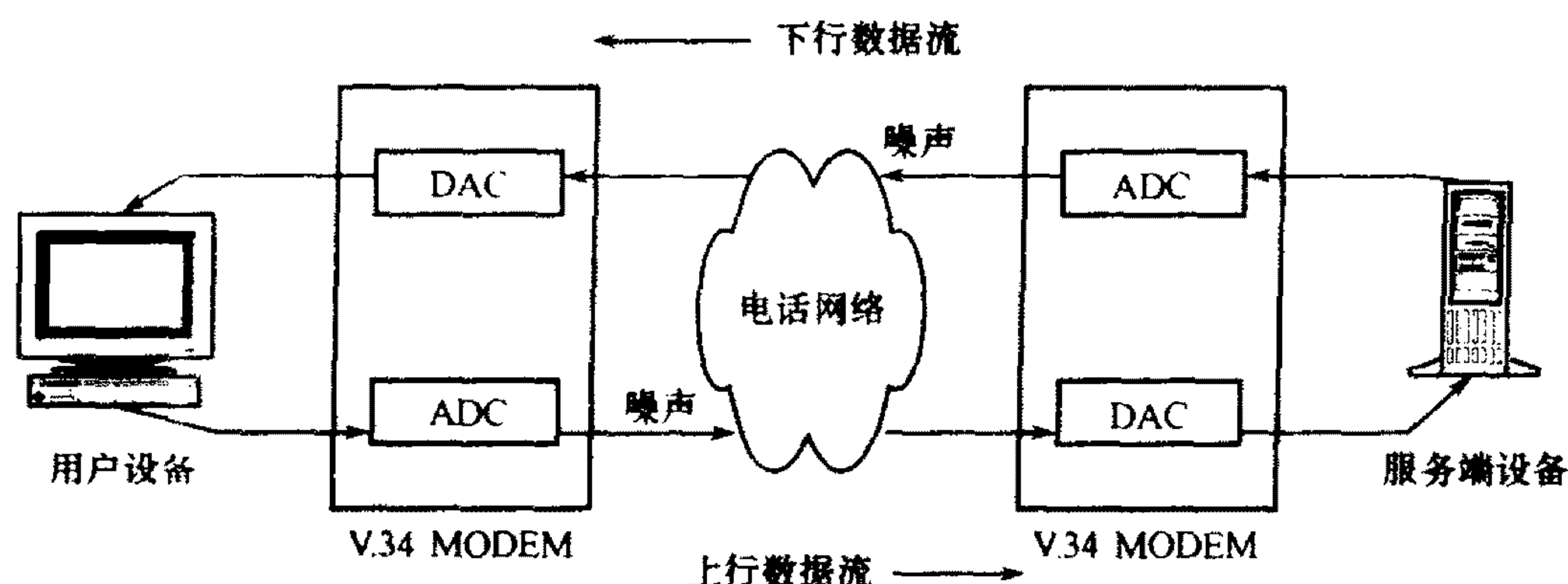


图 9.12 传统调制解调器的传输过程

电话交换网的设计初衷就是用于话音通信的。电话网络工程师想方设法将每路电话的带宽减少,以增加可同时通话的话路数,因而人为地将话音频谱限制在与人类说话相关的频率范围之内,也就是 300 Hz ~ 3 400 Hz。在电话交换网中,为了将模拟信号转变为数字信号,必须对电话信号进行采样。采样频率应该大于电话线上传输的最大声音频率的两倍,即 $2 \times 3\,400\text{ Hz}$ 。为了方便起见,采样频率定为 8 kHz,这样就可以从采样脉冲信号中无失真地恢复出原来的话音信号(以人耳是否能区别为标准)。采样系统使用了 256 级离散 8 位 PCM 编码。由于模拟波形是连续的,而二进制数值是离散的,所以这些采样的数据在电话交换网上传输到另一端再现时,得到的只能是与原来的模拟波形相似的波形。原始的波形与再现后的量化波形之间的差别,就是所谓的量化噪声。由于存在着这个量化噪声,根据仙农理论,使得调制解调器的传输速率限制在 35 kb/s 以下。

但我们知道,量化噪声只影响模数转换,而不影响数模转换。这也是 56 kb/s 技术的关键所在:如果在网络服务方(ISP、联机服务商或公司 LAN)一端的服务器直接与电话交换网以数字干线相连,那么网络服务方发送的数字信号就可以回避带来量化噪声的模数转换,直接到达客户调制解调器的接收端,而不会有任何的信息丢失,如图 9.13 所示,56 kb/s 技术正是借此实现高速传输的。

采用 56 kb/s 技术的服务器端调制解调器采用与电话网络采样频率(8 kHz)相同的频率发送数据(每次 8 位,即 256 个电平),数据经由 PSTN 网络传输到客户端的 DAC(数模转换器)。56 kb/s 客户调制解调器的任务就是鉴别这 256 个电平,每秒恢复 8 000 个 PCM 代码。对于客户调制解调器来说,要恢复服务器调制解调器发送出的 PCM 代码,就必须使其采样时钟与网络编(解)码器的 8 kHz 时钟保持同步。新的 56 kb/s 技术采用了模拟调制解调器中类似的技术来恢复网络时钟。

可以看出,在上述情况下从服务器(采用数字干线与电话交换网相连)向客户调制解调

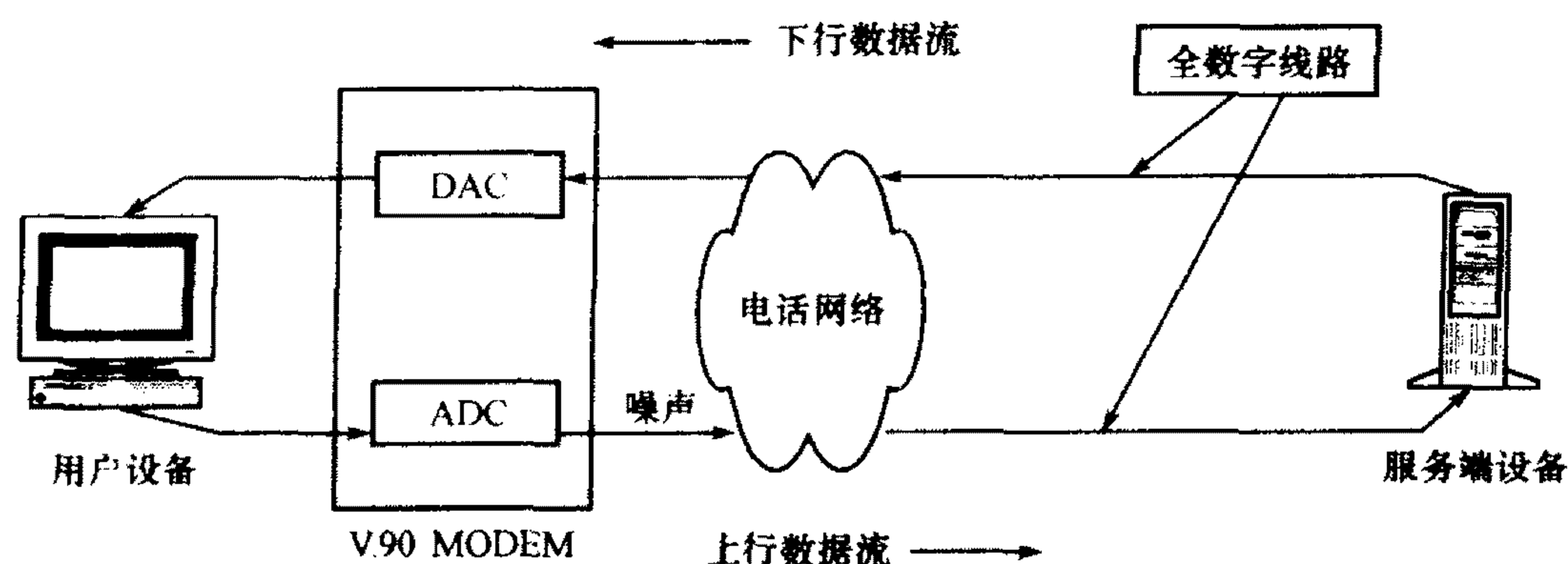


图 9.13 56 kb/s 调制解调器的传输过程

器(采用模拟电话线与电话交换网相连)发送数据,理想的传输速率(下传速率)应该是 64 kb/s($8\,000$ 次/秒 $\times 8$ 比特/次)。但是在实际应用中,线路的原因迫使这个速率必须下调。例如,电话线的传输速率过快,势必会产生较强的电磁辐射,影响到电缆中其他线路的工作。根据 FCC 的规定,56 kb/s 调制解调器技术在实际应用中的工作速率应低于 53 kb/s。

从图 9.13 中可以看出,从 56 kb/s 的服务器端调制解调器发出下行数据由于没有模数转换,可以实现 56 kb/s 的传输速率。但从客户调制解调器向服务器调制解调器发送数据时,仍需经过模数转换,所以无法实现 56 kb/s 的传输速率。在这种情况下,客户调制解调器的上行速率仍然遵循 V.34(28.8 kb/s 或 33.6 kb/s)标准。因此说,56 kb/s 技术是一种非对称的工作方式,即上行(发送数据)的速率最高为 33.6 kb/s,仅下行速率可以实现 56 kb/s。由于在实际应用中用得最多的还是下载数据,对上传速率的要求并不明显,所以这种非对称的 56 kb/s 技术也不会给用户带来不方便。

从以上介绍可知,要想在客户端的调制解调器和服务器端的调制解调器之间建立 56 kb/s 连接,必须满足两个条件:一是服务器端的调制解调器(即 ISP,联机服务商或公司 LAN 一方)必须与电话交换网实现数字干线连接,二是服务器端的调制解调器和客户端的调制解调器均需支持 56 kb/s 技术。

56 kb/s 技术的实现不算复杂,近年来才开发的原因是因为以前的 ISP 很少与电话交换网采用数字干线连接,而是用中继线连接,这样就无法回避模数转换。不具备实现 56 kb/s 技术的前提,自然也就无法提出这方面的需求。

56 kb/s 技术以非对称的方式巧妙地在模拟线路上实现了准数字高速通信,在当今因特网快速发展的时代,迎合了广大用户对高速上网的需求。

现在,随着信息技术的发展,宽带网已开始进入家庭。

4. 调制解调器的使用

传统调制解调器按其安装在计算机内还是安装在计算机外,分为内置式调制解调器和

外置式调制解调器。笔记本电脑使用 PCMCIA 调制解调器。PCMCIA 调制解调器可以当作内置式调制解调器处理。调制解调器按其最大传输速率可分为 56 kb/s、33.6 kb/s、28.8 kb/s 等几种。

内置式调制解调器是一块单独的、直接插入 PC 总线扩展槽内的电路板。内置式调制解调器按其总线接口标准可分为 PCI 卡和 ISA 卡,分别插在计算机的 PCI 插槽或 ISA 插槽中。

内置式调制解调器的连接很简单,使用带有电话机插头的连线,将内置式调制解调器的“WALL”端连接到墙上的电话线插座上,再把带插头的连线将电话机连接到调制解调器的“PHONE”端即可。这样连接的好处是:不使用调制解调器时,可正常使用电话。而调制解调器工作时,电话自动被断开,防止了数据通信被电话干扰。

外置式调制解调器的连接较复杂些。第一步,用带插头的连线将调制解调器的“WALL”端连接到墙上的电话插座上,再把电话机连接到调制解调器的“PHONE”端;第二步,将串行接口连接线的连线插头(25 针)一端接到调制解调器上,另一端(9 针)插头接到计算机的串口 1 或串口 2。最后要为调制解调器接上电源。

无论是内置式调制解调器还是外置式调制解调器,硬件安装好后都还必须在计算机上安装驱动程序和设置拨号网络,这方面的内容请参考相关资料。

9.2 设备驱动程序

9.2.1 设备驱动程序的一般概念

设备驱动程序是操作系统中用于控制外部设备的一种程序模块,用于将连接到主机的不同外部设备的特性和操作系统的高层隔离开。在当前流行的几乎所有的操作系统中(包括所有异化的 UNIX 系统在内),设备驱动程序都被认为是最核心的一种部件,处于操作系统的最深层,故要替换或扩展这些驱动程序是很困难的。

对初涉计算机的人,可能会遇到这样的现象:将光盘放置光驱中,机器却找不到光驱,这是什么原因呢?其实原因很简单,就是光驱驱动程序没有安装。平时使用计算机,不仅仅只是光驱需要安装驱动程序,还有如声卡、显卡、解压卡、网卡、MODEM 和激光/喷墨打印机等都需要安装驱动程序。实际上,计算机中所有的硬件都需要驱动程序,但我们常用的外设如键盘、鼠标、软驱和硬盘为什么未安装驱动程序就能使用呢?这是因为这些设备的接口规范已经标准化了,以至于不需再作任何修改就能在各种环境下使用,所以这部分设备的驱动程序就被固化在 BIOS 中作为标准的驱动程序供 DOS 或应用程序使用(也可以说它们在微

型计算机生产过程中已经被预安装到了系统中)。

从以上的叙述可以看出,所谓设备驱动程序就是对连接到计算机系统的设备进行控制驱动、以使其正常工作的一种软件。驱动程序的作用是通过一组预先定义好的软件接口为操作系统或应用程序提供控制硬件的能力。它的好处有两点:一是由于有了驱动程序这一软件层次,使操作系统或应用程序就没有必要关心硬件设备的具体操作细节,大大降低了软件的开发难度和软件的复杂程度;二是增强了软件的兼容性,例如更换了设备后,只要相应地更换驱动程序即可,而无需将整个操作系统或应用程序都换掉。当然,如果在应用程序中不通过设备驱动程序而直接访问硬件也是可以的,但这会带来兼容性问题,也就是说硬件变化后必须要重新编写全部应用程序。

由于不同的操作系统对硬件的管理、控制、使用的方式方法都存在一定的差异,所以,即使是同一台硬件设备,当其在不同的操作系统中使用时,也需要各个系统中专门设计的驱动程序来支持。因此,在硬件使用前,查找硬件附带的驱动程序,查阅相关驱动程序的安装和配置方法是一项十分重要的工作。

在计算机应用过程中,要掌握计算机硬件驱动程序的一般应用虽然不是一件太难的事,但也存在一些需要澄清的概念和问题,本节将主要讨论在当前计算机硬件新产品不断涌现的情况下,如何在不同的操作系统中应用这些产品,并使其性能得以充分发挥。

9.2.2 Windows 9X 设备驱动程序

Windows 95 以前的系统对于硬件设备的访问一般是通过设备驱动程序进行,客户通过设备驱动程序来获得硬件的参数或者设置。由于 16 位的操作系统基于原来的 DOS,所以客户程序仍然可以通过一些 BIOS 或者 DOS 的 DPMI 中断调用甚至直接访问来实现对硬件的操作。

32 位的操作系统如 Windows 95/98 和 Windows NT 不再基于 16 位 DOS,所以用户如果需要实现对硬件中断、DMA、I/O 或者是绝对存储器访问,都只能惟一地通过设备驱动程序。Windows 95/98 操作系统能够实现多线程、多进程的应用,系统通过一个虚拟机管理器 VMM32、VXD(Virtual Machine Manager)和其他的设备驱动程序合作,来实现多个进程间的协调工作,防止一个进程的运行导致另一个进程的崩溃。

Windows 95 设备驱动程序是用来管理系统资源(硬件或者软件)的可执行二进制代码。通过设备驱动程序,多个进程可以同时使用系统软硬件资源,从而可以实现多进程并行运行。Windows 95 设备驱动程序一般以“.VxD"为后缀名,其意思是虚拟化的设备(Virtual Device)。在一般概念上,设备驱动程序和虚拟设备是同义的,下面的叙述将同时兼用这两个词汇。

Windows 95 是继承 Windows 3.x 而来,所以两者的设备驱动程序采取的是同一种模式,

也就是说,一般为 Windows 3.x 写的设备驱动程序,可以不加改动地运行于 Windows 95 下。但是也有一些区别,一般 Windows 95 下面的驱动程序是以“.VxD”为后缀名,而 Windows 3.x 是以“386”为后缀名。Windows 3.x 的设备驱动程序必须在 Windows 启动时静态载入,而 Windows 95 的设备驱动程序可以在程序运行过程中动态载入。但 Windows NT 采用的是另外一种完全不同的模式,所以 Windows 95 和 Windows 3.x 下的设备驱动程序不能用于 Windows NT。

386 以上的微处理器有 4 个优先级别:0 级、1 级、2 级、3 级,一般操作系统运行于优先级 0 级上,而用户程序运行在 3 级上。VxD 运行于 0 级上,其在内存中的位置也是处在操作系统保护的空間之內的。另外,Windows 还提供一些运行在优先级 3 上的驱动程序,主要是串行口的通信程序和并行口的打印机程序,这些程序以“.DRV”为后缀名。但是一般说来,运行于优先级别 3 上面的 I/O 驱动比运行于 0 级的慢。

1. Windows 9X 和硬件设备的“即插即用”概念

微软公司在开发 Windows 95 时,为解决用户对外部设备硬件参数设置的困扰而开发了一项新的功能:即插即用(Plug & Play, PnP)。这是一项用于自动处理 PC 机硬件设备安装的工业标准,由 Intel 和 Microsoft 联合制定。

用户需要安装新的硬件时,往往要考虑到该设备所使用的 DMA 和 IRQ 资源,以避免设备之间因竞争而出现冲突。这是一项很麻烦的工作,因为有了“即插即用”(PnP),就使得硬件设备的安装大大简化了,用户无须再选择如何跳线,也不必使用软件配置程序,这一切都由操作系统代做了。但要做到“即插即用”,对所安装的硬件就有一定的要求,即必须是符合 PnP 规范的,否则无法做到即插即用。即插即用是 Windows 95 及以后的操作系统最显著特征之一,基于 Intel 体系结构的其他微型计算机操作系统目前尚不具备该特性。

即插即用特性还需要主板具有 PnP 功能,这样在系统启动时由 BIOS 自动读取提供具有 PnP 功能的接口卡的设定参数,自动分配各项资源,并将分配后的设定参数存入主机板中的闪存(Flash Memory),再由操作系统从主板闪存读取编排后的 PnP 界面卡相关设定参数,这样就可以避免以往因 I/O 地址相互冲突所造成的困扰,使整个系统在执行各种程序时有效地发挥系统功能。

即插即用计算机系统的具体内容包括:

① Plug & Play BIOS 提供基本指令集,用于确定在系统开机自检(POST)时所需要的最基本设备。这些设备至少包括显示器、键盘、磁盘驱动器等。

② “即插即用”操作系统 Windows 95 是第一个支持 PnP 的操作系统,之后的 Windows 9X 系统也都支持即插即用。

③ “即插即用”硬件 指由 PnP 操作系统自动配置的一组 PC 硬件设备。PnP 也同样支持打印机、调制解调器、串行口和并行口等,基于 ISA 和 EISA 的适配卡则需要进行适当的修改。

④ “即插即用”设备驱动程序 Microsoft 提供的设备驱动程序支持基本 PnP 设备,例如 IDE 硬盘、CD - ROM 等。

2. 如何解决 Windows 9X 设备的中断和地址冲突

虽然 Windows 9X 支持即插即用,能够自动识别有关设备(或接口)并安装相应的驱动程序,无需人工调整诸如中断 IRQ、I/O 地址、DMA 通道等参数,但在实际安装时,由于即插即用设备品种规格越来越多,新设备层出不穷,再加上有些设备并未完全按照即插即用的规范要求生产,致使系统常常不能正确检测出其参数。特别是在安装设备较多的情况下,很容易引起设备冲突。设备冲突引起的故障类型及表现很多,轻则会使某些设备不能正常运转,重则造成整个系统瘫痪。下面从常见的设备冲突故障表现入手,探讨解决问题的基本思路。

在多媒体设备中,常见的设备有声卡、FAX 卡、网卡、图像处理卡等。在这些设备出厂时,一般都进行了缺省配置。常见设备使用的缺省中断类型号见表 9 - 3,如声卡常用的中断是 IRQ5,MODEM 的中断一般是 IRQ3 或 IRQ4,网卡中断为 IRQ3,解压卡中断为 IRQ10 等等,当然也可以通过硬件和软件的方法来修改这些缺省的配置,尤其是在设备发生冲突时。

表 9 - 3 常见设备中断列表

IRQ	默认设备	可以被其他设备使用否
0	定时器	不可
1	键盘	不可
2	级联到第 2 个中断控制器	不可
3	COM2(或网卡)	也可为 COM4,但两者只能用一个
4	COM1	也可为 COM3,但两者只能用一个
5	LPT2	一般只有一个并口,所以可用
6	软盘控制器	不可
7	LPT1	不可
8	时钟	不可
9	-	可用
10	-	可用
11	常用于显示卡	不可
12	PS2 鼠标接口	如无 PS2 鼠标,则可用
13	协处理器	不可
14	IDE1	不可
15	IDE2	如只用 IDE1,则可用

引起设备冲突的原因大致有以下两个：

① 硬件生产厂家很多,虽然在生产时有一些不成文的标准,但实际上某些厂商并没有真正照着去做,所以两种甚至两种以上设备可能占有同样的中断号 IRQ、I/O 地址或 DMA 通道。

② Windows 9X 不能很好地识别。在 PnP 智能识别方面,Windows 9X 还有一定差距,只要一个设备识别错误,就容易造成连锁反应,导致系统混乱,这种情况尤其是在设备较多时更容易发生。

设备冲突引起的故障现象有以下一些:

① 显示卡与其他设备的中断冲突导致显示卡不能正常使用,现象是开机显示正常,进入 Windows 9X 画面后黑屏,虽然硬盘指示灯亮,但不久后会出现死机。

② COM 口鼠标一般不会与其他设备冲突,但 PS2 鼠标使用的中断为 IRQ12,有时会与其他设备冲突,引起鼠标不能工作。

③ 网卡与 MODEM 冲突,常见 NE2000 兼容网卡的缺省中断为 3,与 COM2 口及有些 MODEM 冲突,所以此时 MODEM 和网卡要么只能使用其中之一,要么改变网卡的缺省中断。

④ 解压卡冲突,常见为中断或 DMA 通道冲突,如若解压卡使用中断 IRQ12,则与 PS/2 鼠标冲突,现象通常为在播放数幅图像后死机。

关于 Windows 9X 的资源冲突问题,可按以下的思路及步骤解决:

① 尽量节省资源,在 CMOS 中关闭不用的设备 如关闭 COM2,可以节省出 IRQ3;

② 按一定的顺序安装设备 由于在安装 Windows 95 时,大部分设备都配有驱动程序,因此在安装时,应该尽量首先安装易引起系统崩溃的设备,比如主板驱动程序、显示卡驱动程序、硬盘驱动程序等,而将 FAX 卡、声卡、网卡等次要设备待主要设备安装完后再进行安装;

③ 尽量采用缺省设置 绝大部分情况下,采用缺省配置是最可靠的方法。所以只要不发生冲突,就无需改变缺省配置。确实需要调整时要仔细阅读该设备的说明书,调整方法一般有修改跳线与软件调整两种;

④ 必要时关闭冲突的设备 冲突发生后,只要系统不瘫痪,就很容易检查系统资源状况,分析冲突原因。方法是在控制面板中选择“系统”→“设备管理”,然后观察各设备的状态。一般不能使用的设备在图标旁边都有明显记号。仔细检查每种设备资源,必要时将使用正常的设备关闭。比如:由于冲突,声卡不能使用,MODEM 可以使用,但 MODEM 使用的中断为声卡的缺省中断 IRQ5,则应将 MODEM 强行关闭。方法是选择 MODEM 后,选择属性,点中“在该硬件配置中禁用”,然后重新启动机器,安装声卡驱动程序;

⑤ 必要时可先拔掉有关板卡 先拔掉引起冲突的接口卡,将其他设备安装完毕后,再插回接口卡并安装该卡的驱动程序,绝大多数情况下不会再发生冲突,虽然此方法较麻烦,但非常有效。

3. Windows 9X 的设备驱动程序的安装

进行 Windows 9X 的设备驱动程序的安装前,首先需准备好要安装的硬件设备和驱动程序(如果没有最新的驱动程序,可从 Windows 9X 的设备驱动程序中选择)及 Windows 9X 的系统安装程序(如 Windows 9X 的光盘、在本地硬盘上准备好 Windows 9X 的光盘备份或与安装有 Windows 9X 的光盘备份网络服务器上建立连接等),因为许多设备在安装时,除了安装设备低层的驱动程序之外,还要安装其他相关的应用程序和软件包,这方面以网络接口卡和调制解调器的安装最为典型。以下是 PnP 设备和非 PnP 设备的安装步骤:

(1) 安装“即插即用”设备

- ① 关闭计算机;
- ② 根据生产商的说明将设备连接到计算机上;
- ③ 打开计算机并启动 Windows。Windows 将自动检测新的“即插即用”设备,并安装所需的软件。

如果 Windows 没有检测到新的“即插即用”设备,则设备本身没有正常工作、没有正确安装或者根本没有安装。请不要使用“添加新硬件向导”。“添加新硬件向导”不能解决此处所提的任何问题。

(2) 安装非“即插即用”设备

- ① 关闭计算机;
- ② 根据生产商的说明将设备连接到计算机上;
- ③ 单击“开始”,指向“设置”,单击“控制面板”,然后双击“添加新硬件”,也可以打开“添加新硬件向导”;
- ④ 按照屏幕提示操作。

对于非“即插即用”设备,在安装前应根据说明书(或诊断软件)仔细检查和设置设备的中断号和口地址,做好记录,以备使用。

4. Windows 设备驱动程序编写方法简介

设备驱动程序的编写有一定规范,需要程序员对 32 位的汇编语言和 C 语言比较熟悉。一般情况下用汇编语言编写,但也可以用 C 与汇编的混合语言实现。开发设备驱动程序的工具软件包括 Microsoft DDK(Driver Development Kit)、MASM 6.11 和 VC 2.0 以上版本。

通常设备驱动程序由 5 个逻辑段组成,它们分别是: VxD _ CODE、VxD _ DATA、VxD _ ICODE、VxD _ IDATA 和 VxD _ REAL _ INIT。

① VxD _ CODE 是保护模式下的代码段,一般这个段包括设备驱动程序的控制程序、回收函数、服务程序和接口函数,这个段也命名为 _ LTEXT。

② VxD _ DATA 是保护模式下的数据段,一般包括设备驱动程序的描述表以及一些全局

变量,这个段也命名为 `_LDATA`。

③ `VxD _ICODE` 是保护模式下的初始化代码段,包括一些初始化时使用的服务程序,虚拟机管理器(VMM)在初始化结束之后将该段取消,这个段也命名为 `_ITEXT`。

④ `VxD _IDATA` 是保护模式下的初始化数据段,包括一些初始化时使用的数据,虚拟机管理器(VMM)在初始化结束之后也将该段取消,这个段也命名为 `_IDATA`。

⑤ `VxD _REAL INIT` 包括实模式下的初始化数据和代码段,虚拟机管理器(VMM)最先就是装入这个代码段,在进程返回之后这个段被取消,再装入其他代码段,这个段又命名为 `_RTEXT`。

需要注意的是,除了实模式下的初始化数据和代码段外,其他4个段都是32位保护模式下平坦模式(Flat Model)的段。

每个设备驱动程序必须首先声明一个虚拟机的名字、版本号、初始化过程、虚拟机控制程序(相当于程序入口),有些设备驱动程序还可以声明设备标志号和接口函数(API)。虚拟机一般用 `Declare _Virtual _Device` 宏来声明,例如:

```
Declare _Virtual _Device VSAMPLED,4,0,VSAMPLED _Control, \
VSAMPLED _Device _ID, VSAMPLED _Init _Order, \
SAMPLED _V86 _API _Handler, VSAMPLED _PM _API _Handler
```

以上声明了一个样例虚拟机,名字叫做 `VSAMPLED`,版本是4.0,控制程序为 `VSAMPLED _Control`,`VSAMPLED _Device _ID` 声明了虚拟机的标志符,`VSAMPLED _Init _Order` 说明初始化的过程,后面两项说明了支持V86模式和保护模式下的两种接口函数。通过这些声明就可以确定一个惟一的虚拟机,系统就可以将它和别的虚拟机区别开来。

每个虚拟机都需要一个虚拟机控制程序。虚拟机管理器(VMM)通过这个程序来向该虚拟机传递控制信息,系统通过控制消息来控制虚拟机的执行,例如初始化虚拟机、虚拟机状态的改变等工作。

一个虚拟机需要提供一些服务功能,这些功能可以被虚拟机管理器(VMM)或者其他的虚拟机使用。但是,虚拟机不像Windows DLLs一样提供出口函数(Export Functions),虚拟机管理器(VMM)通过0x20号中断来实现与虚拟机的连接,中断的句柄通过一个服务号来决定到底哪个虚拟机支持这个服务功能。

一个虚拟机通过 `BeginProc` 和 `EndProc` 两个宏来定义它的服务功能,与汇编语言相似。例如:

```
BeginProc VSAMPLED _Service _Routine, Service
< Main body of the service... >
EndProc VSAMPLED _Service _Routine
```

一般 `BeginProc` 的名字后面有一个参数: `Service` 或者 `Async _Service`,后者是说明这个服

务程序可以被异步地调用,多用在中断服务程序中,异步服务程序必须是可以重入的,所以它不能调用不能重入的服务程序。

9.3 计算机中的多媒体技术

人类接受的信息中约有 80% 来自视觉,是人类最有效和最重要的获得信息的形式。人类最方便的信息交流方式则是“听”和“说”。视觉涉及到文字、图形图像、视频信息,“听”和“说”则涉及到声音信息。而多媒体技术使计算机拥有了处理声音、图形、图像、视频、文字的综合处理能力和人机交互能力,使计算机的应用领域迅速扩展到家庭娱乐、教育、艺术、商业广告等行业。多媒体技术的出现与发展是与计算机技术、网络技术、信息存储技术、外围设备技术的发展紧密联系在一起的。其中包括图像和声音的压缩方法、多媒体信息的网络传输、大容量存储设备的开发、CPU 对多媒体信息处理的支持,多媒体设备的开发、节目源的编辑制作与播放软件等等。

9.3.1 多媒体计算机

多媒体一词来源于英文 Multimedia,其核心意思就是“媒体”。在计算机科学中,“媒体”包括了两个方面的含义:一是信息的物理载体,二是信息的表现形式。物理载体是指存储信息或表现信息的实体,如磁盘片、磁带、打印纸等;而声音、文字、图像等则是信息表现的媒体。计算机学科中的媒体就是指这些表现信息的媒体。现代微型计算机系统不仅能够处理文字、数据这样的信息媒体,还能够处理声音、视频图像、动画等其他不同形式的信息媒体。将具有能够同时对文字、声音、视频图像、动画等多种媒体进行编辑、存储、播放等处理的计算机称为多媒体计算机。

1. 多媒体计算机的组成

多媒体个人计算机的硬件组成如图 9.14 所示。除基本的主机、磁盘驱动器、显示器、网卡等外,主要包括了音频信息处理部件、视频信息处理部件及光盘存储器等。

除硬件系统外,多媒体计算机同样需要软件的支持,其软件环境如图 9.15 所示。图中最底层是各种多媒体硬件设备及控制卡;其上一层为音频和视频信息的压缩和解压缩,因要求的处理速度较高,这一层也多由硬件完成;多媒体设备的 I/O 控制层是多媒体设备与操作系统的接口,具有一般接口的功能,负责设备的驱动和控制,并提供高层软件调用的接口。

图 9.15 所示的上面 3 层为完全意义上的软件系统。多媒体操作系统是在一般操作系统的基础上加入多媒体资源管理和信息处理功能,为用户提供一个进行进一步开发的平台。

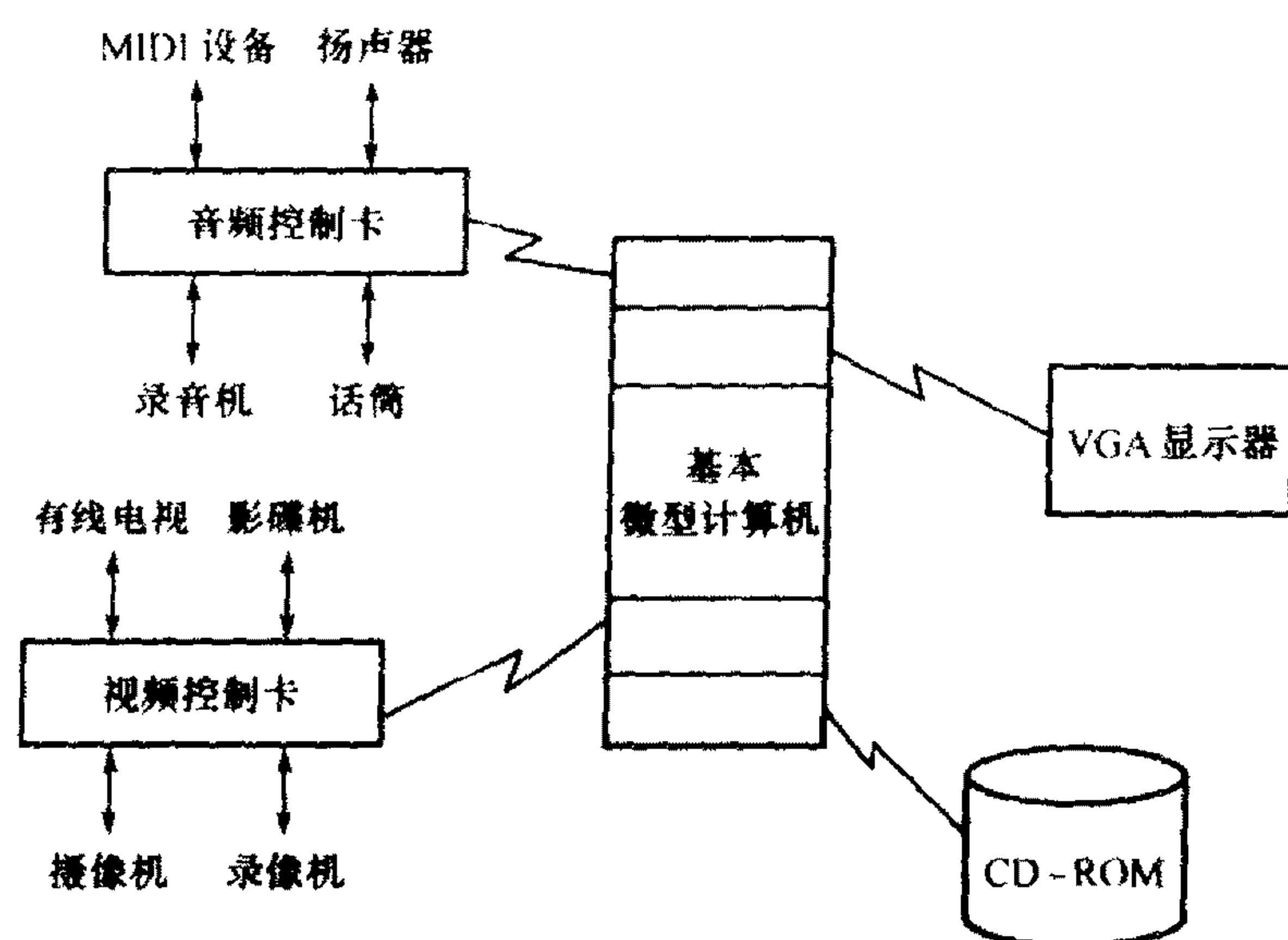


图 9.14 多媒体个人计算机的硬件组成

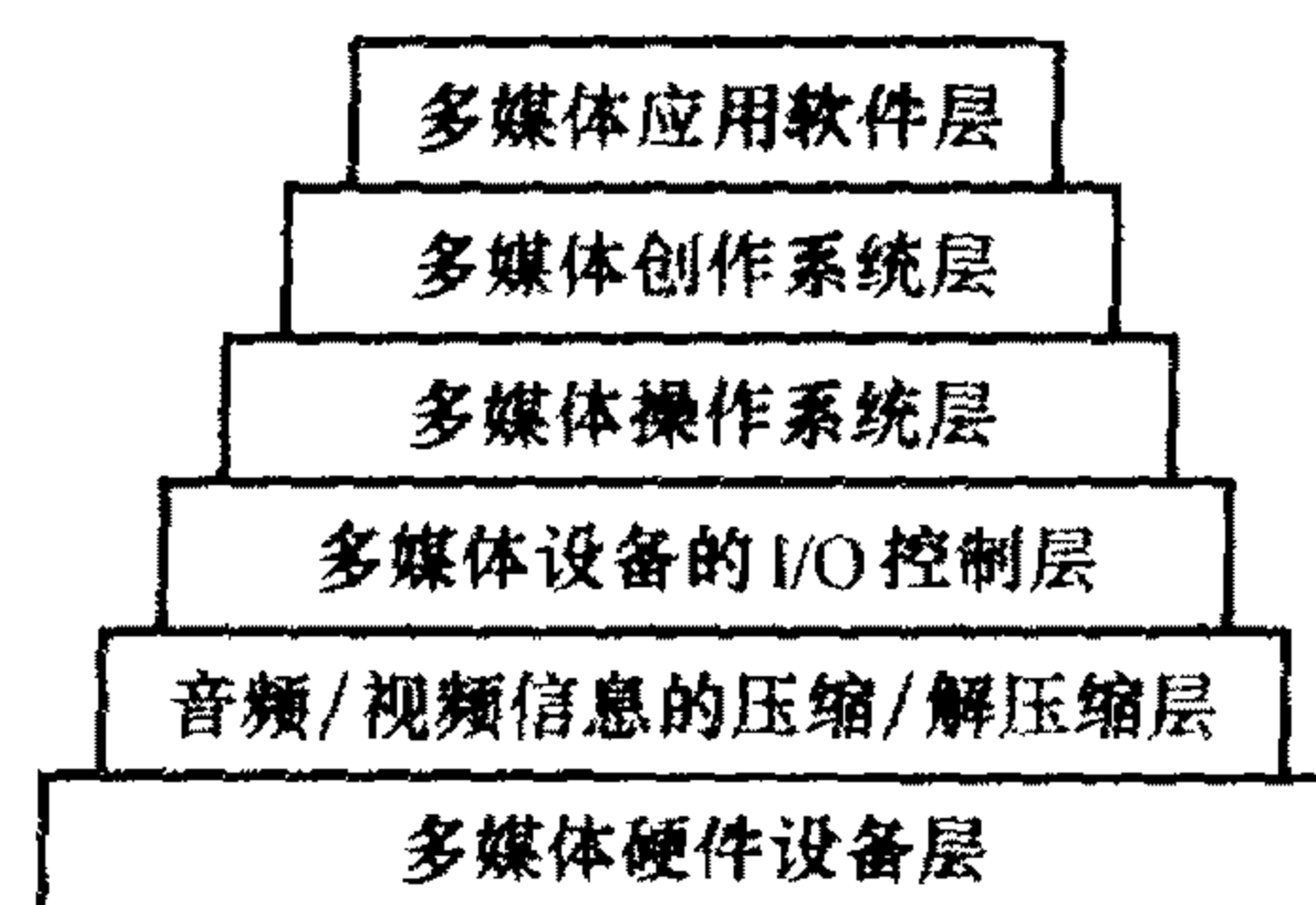


图 9.15 多媒体计算机的软件环境

操作系统的上边是多媒体创作系统层,主要是用于多媒体软件的开发工具,包括编辑工具和写作工具。编辑工具一般除编辑功能外还具有播放的功能,包括如文字处理、彩色图像处理、动画制作、声音编辑等工具软件。写作工具(Authoring Tools)基本上是一种应用程序生成器,用于帮助应用程序开发人员提高工作效率。根据不同的开发类型,写作工具分为脚本式(Script-based)、流程图式(Flowchart-based)和时序图式(Timeline-based)三种。目前常见的多媒体软件写作工具有:Toolbook、Icon Author、Authorware Professional及Action!等。

2. 多媒体个人计算机的标准

多媒体计算机产品主要有两种:多媒体工作站和多媒体PC机。1991年11月,由Philips、Microsoft等14家著名厂商组成了多媒体市场协会,制定了关于多媒体PC机的第一代标准Level I(简称MPC1)。MPC标准随着计算机技术的发展在不断地更新,现在最新的标准是MPC3。

MPC1标准于1991年11月提出,它建立在10 MHz的286AT的基础之上,但很快又修改成了采用16 MHz的386SX。MPC2标准则是1993年制定出来的,这一技术规范定义了第二级别的最小系统功能。

由于计算机和多媒体产品的进一步发展,更名为多媒体计算机工作组的多媒体市场协会1995年6月公布了最新的多媒体计算机标准,即MPC3标准。

这3个标准覆盖的范围如下:

① MPC1标准规定了计算机多媒体的基本要求,提供了多媒体计算机的基本框架;

② MPC2 进一步扩展了多媒体计算机的结构,使多媒体计算机技术逐步标准化;

③ MPC3 并不是用以替换 MPC2 的,它只是对多媒体计算机的表现能力提出了更高的要求。

MPC1、MPC2 和 MPC3 所制定标准的要点的比较见表 9-4。

表 9-4 MPC1、MPC2 和 MPC3 标准的要点比较

要求	MPC1 标准	MPC2 标准	MPC3 标准
CPU	80386SX 以上	25 MHz 80486 以上	75 MHz Pentium 以上
RAM	2 MB 以上	4 MB 以上	8 MB 以上
磁盘	1.44 MB 软驱、 30 MB 硬盘	1.44 MB 软驱、 160 MB 硬盘	1.44 MB 软驱、540 MB 硬盘
CDROM 驱动器	数据传输率 150 KB/s 符合 CD-DA 规格	数据传输率 300 KB/s 平均存取时间 400 ms 符合 CD-XA 规格 具备多段式能力	数据传输率 600 KB/s 平均存取时间 250 ms 符合 CD-XA 规格 具备多段式能力
声频	8 位声卡	16 位声卡 FM 合成器 MIDI 播放器	16 位声卡 波表合成技术 MIDI 播放器
图形 性能	VGA 640 × 480 × 16 色或 320 × 200 × 256 色	Super VGA 640 × 480 × 65535 色在占 40% CPU 时间时显示速度为 1.2 M 像素/秒	可进行颜色空间转换和缩放,视频图像子系统 在允许时可进行直接帧存取,以 15 位/像素, 352 × 240 像素的分辨率,30 帧/秒(或 352 × 288,25 帧/秒)播放视频,不要求缩放和裁剪
视频 播放	无	无	MPEG1 播放(硬件及软件),可进行直接帧存 取,以 15 位/像素,352 × 240 像素的分辨率,30 帧/秒(或 352 × 288,25 帧/秒)播放,视频不要 求缩放和裁剪。所有的 CODEC(编码和解码) 都应在以 15 位/像素,352 × 240 像素的分 辨率,30 帧/秒(或 352 × 288,25 帧/秒)播放视 频时支持同步的音频/视频流,不丢帧
用户 接口	101 键 IBM 兼容 键盘、鼠标	101 键 IBM 兼容 键盘、鼠标	101 键 IBM 兼容键盘、鼠标
I/O	MIDI、游戏杆、 串行口、并行口	MIDI、游戏杆、 串行口、并行口	MIDI、游戏杆、串行口、并行口
系统 软件	Windows 3.0 多媒体扩 展版或 Windows 3.1 或 MS-DOS CD- ROM 扩展版	Windows 3.0 多媒体扩 展版或 Windows 3.1 二进制兼容的系统	Windows 3.x 以上或 DOS 6.xx 或二进制兼容的 系统

9.3.2 多媒体技术概述

1. 图形/图像和视频

1) 多媒体系统的图像来源

多媒体应用中的信息源主要通过以下几种途径得到:

① 直接购置数字化的图像。介质可以是 CD-ROM、磁盘或磁带。

② 使用专门软件创作所需要的图像。Paint、Flash、Paintbrush 等软件都有大致相似的功能。它们有很好的图形用户接口,可通过图符菜单选择,能用鼠标器、输入笔和数字化板描绘各种图形,具有填色、剪贴、缩放、平移、旋转、调色板设置等各种功能。利用这些软件可以生成屏幕上的图符、动画等。

③ 用彩色扫描仪将照片或艺术作品扫描后得到数字的图像,可以将照片、艺术作品转换成全彩色的位图图像。

④ 利用电视摄像机来捕获图像。摄像机与插在计算机内的视频采集卡相连接。视频采集卡又称为帧捕获器,其作用是把来自摄像机的模拟信号转换成数字信号。摄像输入设备与扫描仪之间最大的差别是,扫描仪只能输入平面的图像,而摄像机可以捕获三维空间景物。摄像机输入平面的图像时,速度也比扫描仪快。

2) 图像的输入设备

计算机输入图像的设备有扫描仪、电视摄像机、光盘机和磁带放像机。

(1) 扫描仪

扫描仪主要用于将平面图形转换成数字信息。其主要技术指标有分辨率和灰度等级。无论是正片还是负片扫描仪,分辨率都用点数每英寸(dpi)来表示。分辨率越高,识别最小细节的能力越强,产生的图像越清晰。单色扫描仪的灰度等级是指识别和反映像素点明暗程度的能力。如果每个像素点用 8bit 编码,则能反映 256 个灰度等级。灰度等级越高,产生的图像越逼真。彩色扫描仪扫描图像时,要对像素点分色,把一个像素点分解为红(R)、绿(G)、蓝(B)基色的组合。对每一个基色的深浅程度也用灰度等级表示,这就是彩色精度。高档扫描仪对每一基色可识别和表达 4 096(12bit)级灰度,处理时取每色 8bit,能确保 16 777 216种彩色再现,即真色彩。

(2) 数字照相机

数字照相机(Digital Camera)是一种图像获取设备,它使用 CCD(Charge Coupled Device,电荷耦合元件)对光照产生的光电效应来成像。CCD 是一种半导体芯片,由排列成固定阵列的光电管组成,能够把光学影像转化为数字信号。一块 CCD 上包含的像素数越多,其提供的画面分辨率也就越高。CCD 阵列就放在镜头之后。光强度使光电管带电,单个 CCD 光电管

上积累的电荷取决于入射光的强度,光电管的电荷产生电压并随后被输入到模数转换器(A/D)中,转换器把电荷电压转换成数字量,存储在相机内的存储器中,这些电信号就代表了所拍摄的图像。

数字相机最终产生的照片完全是一个二进制的数字文件,可直接传送给计算机进行显示、处理、储存和再输出。具体的输入过程是通过导线把相机和计算机的并行接口或 USB 接口相连,使用厂商提供的专用程序进行下载。

另外,数字相机还可通过视频线在电视上进行幻灯播放或不通过计算机直接打印(但这种输出方式一般只适用于有同一品牌的数字相机和打印机)。

数字相机的内部固化有存储器,能够永久地存储一定数量的照片图像。但它的容量有限,在存满后若要再存入新的就必须先将旧的删除。因此,数字相机可配备多种外部存储设备,如 PCMCIA 卡、Compact Flash 存储卡、SWT 存储卡、存储棒、3.5 寸软盘等,可自由插入相机或取出。

(3) 电视摄像机、录放像机等视频设备

严格地说,电视摄像机不能算作计算机输入设备,它仅是一种将活动图像记录到磁带(或半导体存储器)上的电子设备。录制到磁带上的图像可通过录放像机进行回放。但回放的图像信息为模拟视频信号,需要通过视频捕捉卡转换成数字信号才能由计算机进行存储、处理。黑白电视信号的捕捉过程比较简单,只需对亮度信号进行转换。由场同步信号锁定帧的起止点,由行同步信号确定行的起止点,然后经过采样、量化和编码等步骤就能将电视信号数字化。对于彩色电视信号则还要考虑到不同的制式,如 NTSC、PAL 和 SECAM 等,并且还要将色差信号 YUV 转换为基色信号 RGB,以便计算机处理和显示。

3) 图像的采集和存储

输入的图像可分为静态图像和动态图像。静态图像的输入不用过多地考虑速度问题,只需考虑分辨率、彩色精度和数字化后的数据量,对于动态图像,则要着重考虑图像的数字化速度和存储数据量问题。

以印刷品形式出现的静态图像可用扫描仪或电视摄像机输入。静止实景可先拍成照片,然后用扫描仪输入,或直接用电视摄像机拍摄。

如果仅仅捕捉单帧电视画面,可以将电视画面的捕捉当作静态图像信号的采集来看待,不必过多地考虑整幅画面的输入速度。此时整幅画面可以一次扫描输入,也可以多次扫描输入。由于电视画面更新速度很快,如 PAL 制式为每帧 1/25 秒,隔行扫描,因而 A/D 转换和数据输入的速度都必须很高。静态画面可用多次扫描,拼图合成的方法输入整幅画面,例如在一帧的显示时间内只输入一行,这样就可以用低速 A/D 和低速输入接口来实现单帧画面的捕捉。使系统的难度和造价降低。对于动态图像的输入,捕捉卡的处理速度必须与电视信号匹配,1/25s 完成一帧 PAL 制式电视画面的数字化和输入。1/30s 完成一帧 NTSC 制

式的电视画面的输入。

数字化的图像产生大量的数据,尤其是动态图像,数字化后的数据量大得惊人,这给图像存储带来了困难。尽管光存储技术已使计算机外存容量激增,但与图像的数据量相比,存储容量仍相差甚远。例如,以 320×240 的扫描像素和 24bit 彩色来量化动态电视画面,每分钟产生的数据为:

$$(320 \times 240 \times 24b \times 25 \text{ 帧} \times 60s) / 8 \approx 345.6 \text{ MB}$$

目前一张 CDROM 光盘的容量为 650 MB,这样一片光盘只能存储不到 2 分钟的视频节目。很明显,若不采取措施,直接将图像信息进行存储几乎没有什么实用价值。

降低图像数据量的最简单办法就是牺牲图像的分辨率和彩色精度。在多媒体系统中,图像的分辨率和彩色精度要根据应用需求确定。

减少图像数据量的另一个有效措施就是数据压缩。图像数据有很大的冗余度,采用压缩编码技术可以大大降低图像的数据量,好的压缩算法可使压缩率达到 1:100 以上。常见的编码技术有 MPEG1、MPEG3、MPEG4、Real 等。

4) 图像的处理

图像输入到计算机后,应对它进行各种处理,例如对图像实施以滤波、增强、特征抽取等信号处理手段和旋转、缩放、变形、剪裁、着色、与文本和声音混合等编辑手段,以满足各种应用需求。这些工作可使用专用图像和视频处理软件来完成。

5) 图像的输出设备

图像的输出设备一般包括打印机、显示器及录像机、电视机等。

① 打印机 主要类型有喷墨打印机和激光打印机。打印机的技术指标与扫描仪相同,主要有分辨率和彩色精度,分辨率也用 dpi 表示,一般为 200dpi ~ 300dpi。彩色精度最高可达 16 777 216 种彩色。

② 显示器 很多显示器接收 RGB 基色信号,因而可将数字图像信号转换为 RGB 模拟信号,加入同步和消隐信号送往显示器。有些显示器也接收复合视频信号或 S-视频信号,此时就需要将数字化的 RGB 信号转换为模拟的复合视频信号或 S-视频信号。

图像信号经显示卡转变为 VGA 或其他标准的信号,就可以在计算机屏幕上显示。显示卡有 EGA、VGA、SVGA 和 XGA 等多种标准,提供了不同的显示分辨率和彩色精度。每屏的扫描像素 CGA 的为 640×200 , EGA 为 640×350 ,其他为 640×400 , 640×480 , 800×600 , 1024×768 , 1280×1024 不等,彩色数量有 16 色, 256 色, 32768 色、1677 万色等。

③ 录像机和电视机 数字化的图像信号经 D/A 转换, RGB - YUV 转换,并根据录像机的制式(NTSC、PAL 或 SEAM)进行相应的编码,合成全彩色电视信号 FBAS,送入录像机记录,并可在电视机中播放。

2. 音频技术

多媒体系统回放声音的方法,主要有数字音响、CD 光盘播放、通过 MIDI 驱动内置或外置的合成器三种。

1) 数字音响

数字音响是数字技术与音响技术相结合的产物。通过 A/D 转换器将采样到的声音转换为数字信号,播放时,再做 D/A 转换,把数字量转换为波形。这种数字化采样方法能够减少信号的失真度。数字化采样要占用很大的存储空间,采样频率越高、量化级别越多,声音质量越好。一般来说,采样频率高于被采样波形最高频率的二倍以上即可获得满意的音质。由于人耳的听觉上限大约在 20 kHz,因此,采样频率达到 40 kHz 以上即能够达到最佳听觉效果。例如,常见的 CD 音源文件的采样频率即为 44.1 kHz。比特数(位长)也是衡量采样质量的一个参数,它是指每次采样用多少二进制位来表示,8 比特的采样能描绘 $2^8 = 256$ 种变化,16 比特的采样则能描绘 $2^{16} = 65\,536$ 种变化。位数越多,其声音质量就越好。多媒体系统声音采样的比特数有 8 和 16 两种。

对于音频信号的数字化方法,一般采用以下三种:

- ① 脉冲编码调制(PCM);
- ② 差分脉冲编码调制(DPCM);
- ③ 自适应差分编码调制(ADPCM)。

2) CD 光盘播放

多媒体系统一般都配备有 CD-ROM,用它可以直接播放 CD 光盘。CD 光盘上的数字化音乐通常是以压缩形式存放的,常见的格式有 AVI、MP3、MIDI 等。

3) 通过 MIDI 驱动内置或外置的合成器

MIDI 是乐器数字接口(Musical Instrument Digital Interface)的缩写,它产生声音的方法与上述两种有很大的不同,它只发送给合成器一系列指令,具体发出声音的是合成器或音源,而这些合成器或音源本身又有一套对声音进行编程的参数。

早在 1982 年,MIDI 就形成了一个通过电缆将电子音乐设备连接起来的协议。这一协议是向有关设备传送数字化信息的命令。该协议成为设计人员共同遵守的一个标准,使不同厂家生产的电子音乐合成器可以互相发送和接收彼此的音乐数据。

1985 年,软件设计人员又设计出一种程序,它可根据一套作曲算法,用 C、Basic 或 Fortran 语言编程,生成 ASCII 乐谱,并翻译成可在合成器上演奏的序列文件。

1988 年,正式提交给 MIDI 制造商协会的 MIDI 标准,成为数字式音乐的一个国际标准。它规定了电子乐器与计算机之间进行连接的电缆与硬件方面的标准,以及电子乐器之间、电子乐器与计算机之间传送数据的通信协议。

(1) MIDI 的基本配置

所谓 MIDI 设备,是指它包含了处理 MIDI 信息的微型计算机及有关硬件接口。

一台 MIDI 设备可以有 3 个端口,它们分别被称为 MIDI In、MIDI Out、MIDI Thru。它们的作用是:

- ① MIDI In 接收来自其他 MIDI 设备的 MIDI 信息;
 - ② MIDI Out 发送本设备生成的 MIDI 信息;
 - ③ MIDI Thru 将从 MIDI In 端口传来的信息转发到相连的另一台 MIDI 设备上。
- 端口之间的连接电缆可以用 MIDI 统一的屏蔽双绞线,两端带 5 针的阳插头。

(2) MIDI 合成器

合成器是一种电子设备,使用数字信号处理器或其他类型的芯片产生音乐和声音。它产生的波形通过声音发生器送往扬声器。PC 机与 MIDI 合成器之间的接口可以有几种方式,比较典型的一种如图 9.16 所示。

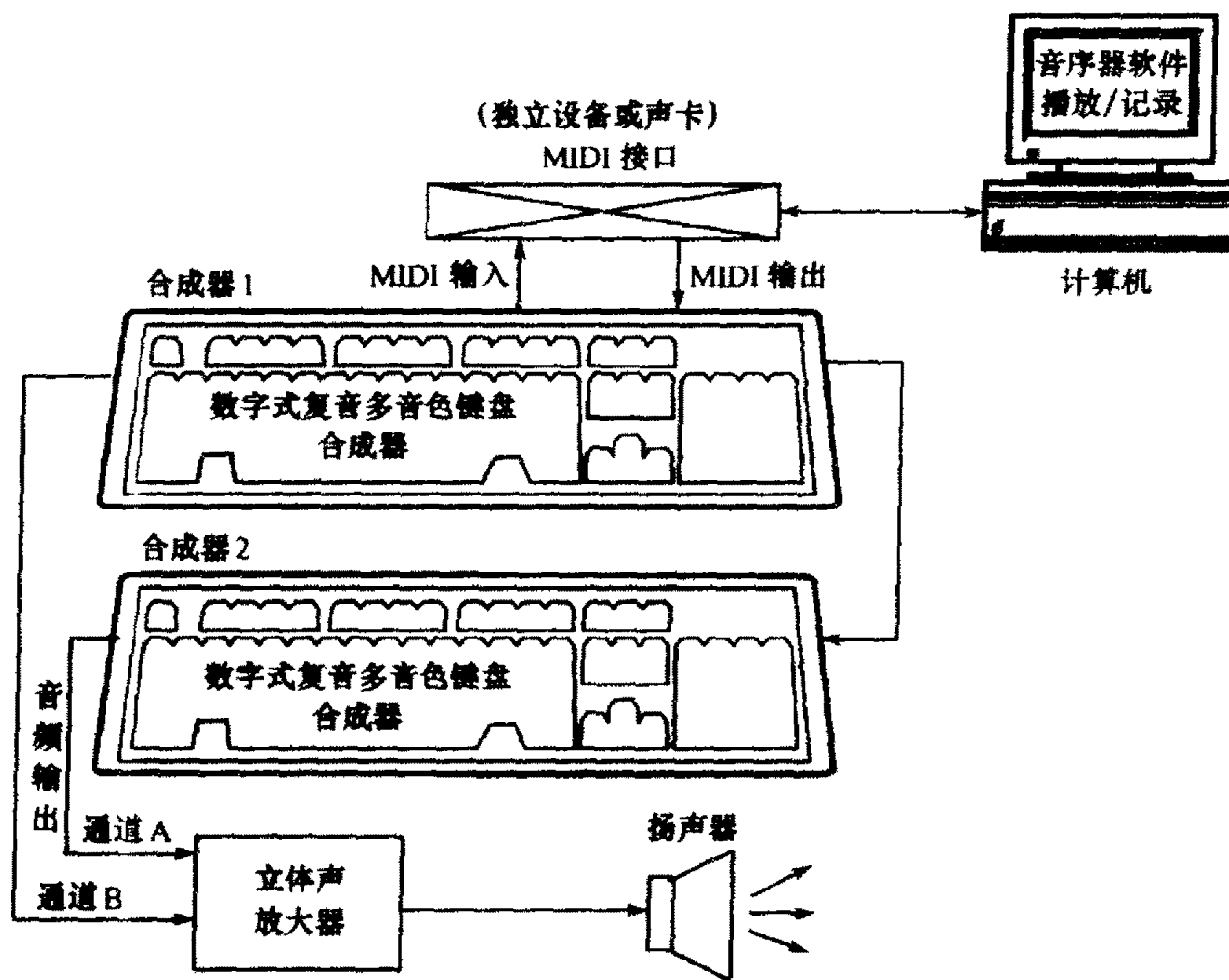


图 9.16 计算机与 MIDI 合成器的接口方法

(3) MIDI 软件音序器

音序器是用于记录、编辑和播放 MIDI 文件的一种软件。

音序器可以帮助专业音乐工作者和音乐爱好者通过 MIDI 文件进行作曲。它也可以帮助计算机作曲 (Computer Music), 用于乐曲修改及播放。

作曲者可以在合成器键盘上弹奏乐曲,当按下键盘上的一个键时,MIDI 电缆就输出 3 个字节的音符起止信息,这些信息表明了哪个键的按下与放开。合成器根据这些信息,也将这个音符演奏出来。计算机将 MIDI 信息存储到存储器中,或者作为 MIDI 文件存入磁盘。这些数据可以进一步加工,或者与其他 MIDI 数据合并。经过编辑后的 MIDI 文件可以送到合成器中再播出来。

计算机作曲软件一般用高级语言编程,按照特别的作曲的算法,设置各种音乐参数(如音高、节、拍、音量、接合、音色……)写出乐曲程序。这样产生的乐曲程序还不能直接在合成器上演奏,必须经过编译,把乐曲程序文件转换成 MIDI 文件,才可以通过音序器作进一步的加工,或在合成器上播放出乐曲。

(4) MIDI 文件

MIDI 信息用一种标准的格式记录,并作为文件而存储,这个文件就称为 MIDI 文件。MIDI 文件实际上就是数字形式的乐谱,它由一系列的乐符组成,还包括每个乐符的键,通道号、音高、音长、音量、键落下的速度,以及乐器的配置等。

3. 多媒体的质量度和带宽问题

多媒体技术中,不同的对象类型具有不同的质量度量标准。例如,图像以像素/英寸来衡量,其分辨率越高,图像质量就越好。对于文本显示,要求有 100 ~ 200 像素/英寸的分辨率就可以了。激光打印机和办公室用复印机提供的质量水平是从 300 像素/英寸到 600 像素/英寸。用于出版书籍的专业胶板印刷质量是从 1 200 ~ 1 800 像素/英寸。

声音质量以采样速率和用于表示幅度的位数来衡量。较高的采样频率能获取较高的保真度。同样,较高的位数能更精确地获得幅度变化。两个因素都对音质有帮助。8 位量化级别,4 kHz 的采样频率对于(单声道)语音级别声音来说是可以接受的最低水平。音乐质量要求用 16 位,8 kHz 的采样频率。对于 CD 质量的立体声,应使用 16 位,44.1 kHz 的采样频率,多通道立体声甚至要求更高的质量。

录像机质量对于视频显示来说被认为是要求的最低水平。大多数录像机实现了在电视机上大约 300 线的指标,相应的显示器扫描像素 320 × 240。

另一个视频质量的度量是用于定义彩色的位数。16 位彩色是基本要求,一般应达到 24 位彩色。

质量的第三个度量是帧每秒。广播电视以 60 帧每秒(隔行)进行显示,它相当于非隔行的 30 帧每秒显示。低质量的视频显示系统也允许以 15 帧每秒进行显示。

从表 9-5 可看到,对传输带宽的要求在不断增加,并使得对压缩的需要变得很突出。此外,对于所有获取或播放多媒体对象的子系统来说,也必须考虑传输带宽的要求。

表 9-5 多媒体对象的质量和传输带宽

对象类型	文本	图像	音频	视频
对象描述	代码: ASCII, EBCDIC	位图图形、静止图 片、传真	数字化语音或音频 编码流	数字电视图像
对象质量 度量	2 KB/页	未压缩: 300 像素/英 寸时为 7.5 MB/页 压缩后: 75 KB/页	语音/电话: 8 kHz, 8 位(单声道)音频 CD: 44.1 kHz, 16 位	中级 640 × 480 像素 × 24 位色 × 30 帧/秒
传输带宽	16 kb/s	600 kb/s	704 kb/s	220 Mb/s, 未压缩; 1.5 Mb/s, 压缩

4. 解压缩技术

当多媒体数据对象,如二值(黑白)文档图像、灰度图像、彩色图像、摄影或视频图像、音频数据或语音数据对象、动画图像及全运动视频等被数字化后,就产生了大量的数字化的数据。确切的数据量由采样频率、量化级别、彩色位数和图像分辨率决定。图像分辨率的提高将会使对象数据量以几何级数增长。

非常大的数据对象要求极大的数据存储空间。随着数据存储量的增加,用于检索数据的存储时间就会增加。光媒体能够在较小的空间存储大量的数据,但它比磁记录媒体的速度慢,另外,在网络中维护大量现场数据会带来新的问题。

动态对象对数据的传送速度有很高的要求。虽然网络速度已在持续地增长,目前已可高达 1 000 Mb/s,但极大容量的数据对象仍要好几秒钟来传输,尤其是当多个视频流同时传输时问题更加严重。

通过以上分析可知,提高多媒体信息的存储和传输效率对多媒体系统来说至关重要的。为了使多媒体数据的存储和传输达到实用的要求,多媒体中的数据对象必须进行压缩处理,以减少存储与实时传送的数据量。

多媒体对象的压缩标准包括无损压缩及有损压缩两种。

1) 无损压缩

无损压缩标准是为保留原始多媒体对象——图像、语音/音频或视频中的所有信息而设计的。其主要思想就是设法消除数据模式中的冗余度。ITU 的 Group3、4(G3、G4)标准是无损的。

ITU Group3 1D 压缩方案也称为游程编码,这一方案基于这样的假设:一个典型扫描线具有由相同颜色(黑或白)像素组成的长游程。压缩后的数据只包括像素值和重复次数。注意这种方案只是为黑白(二值)图像设计的,而不是为灰度或彩色图像设计的。其基本应用是传真系统。由于它非常简单,这一方案至今还在继续用于传真上。但对正规的文档图像

系统就不适用了,因为即使压缩之后,图像容量仍很大。

ITU Group3 2D 压缩方案也称改进式游程编码。这一方案更常用于基于软件的文档成像系统,它提供了相当高的压缩比,用软件解压缩也比 ITU G4 标准更加简单。其压缩率平均在 10~25 之间,位于 ITU Group3 1D 与 ITU Group4 的中间水平。这一方案采用了改进式 READ(Relative Element Address Designated,相对元素地址指定)算法。它将一维编码方案与二维编码方案结合在一起。另外,它是以图像的统计特征为基础的,即基于这样一个事实:横跨相邻扫描线的图像数据是冗余的。如果在某一指定扫描线上存在黑白过渡,那么在下一扫描行向前或向后 3 个像素的位置上,也会发生黑白过渡,文本中的一行会有 20~30 条扫描线,具体数字由扫描分辨率决定。该行中的许多条线都会有相同的黑色像素和白色像素区间,这是由字符的轮廓决定的。因此需要存储的信息只是字符中轮廓的变化,即连续线的变化。虽然用这种方法编码的图像不如 ITU Group4 编码法的压缩率高,但解码(通常在软件中进行)却比 Group4 快得多。

ITU Group4 压缩是二维编码方法。在这一方法中,第 1 条参考线是一条想像的全白线,位于图像顶部之上。使用这条想像的白线对第 1 组相同的像素进行编码。这就成为下一条扫描线(现行编码线)的参考线。新的编码线又成为下一条编码线的参考线,以此类推。每条连续线都成为下一条线的参考线。

虽然用软件进行压缩和解压缩也是可行的,但通常这种编码还是基于硬件的方法,以获得良好的效果。这种编码法的压缩效果非常好,压缩率可高达 35%,但是一条线中的单个位错误会导致整个图像颜色的颠倒。

ITU Group4 标准只是为解决黑白图像的压缩问题而设计的。二维编码可以进行横向或纵向压缩,但不能处理像素的影调或彩色图像中像素的色彩问题。例如,一幅 64 阶灰度图像要求存储每个像素的影调信息。由于一个像素到下一个像素之间(或者,更可能的情况是一组相同影调或颜色的像素到下一组之间)影调颜色发生了变化,压缩的程度就因为这种变化而降低了,新的影调信息也必须存储。另外,影调的变化比从黑到白或白到黑的变化发生的频率要高许多。

2) 有损压缩

由于人眼的视觉特性,对静态的高分辨率图像的分辨率损失不会产生引人注意的不利效果。而对运动图像来说,人眼的注意力对图像中的变化比较敏感,对不变化的地方则不太敏感。另外对大面积的相同色彩也不太敏感。根据这些特点,产生了不同的图像压缩标准,其中典型的有 JPEG 和 MPEG。

由 ISO 和 ITU 工作委员会共同制定的 JPEG 标准是为静止彩色图像和灰度图像(又称连续色调图像)进行压缩的标准化规格。其原理就是在可接受的范围内降低图像的分辨率,以减少图像的信息量。它可将图像数据压缩至原始图像的 1/10~1/30,而其回放质量仍保持

在可接受的范围内。

另一个标准 MPEG 则是专门为运动图像压缩为目标的标准。这个用于视频的压缩算法首先是由 CCITT 现为 ITU 为电话会议和可视电话设计的。用于这种标准的数字存储媒体包括数字音频带(DAT)、CD-ROM、可写光盘(包括 WORM)、磁带和磁盘,以及局域网(LAN)和 ISDN 广域无线通信所用的通信通道。

与静止图像压缩不同,全运动图像压缩与时间密切相关。其压缩程度用特定分辨率下的传输率(如 150 kb/s)来描述。MPEG 可用 1.15 Mb/s 的速度传送彩色运动图像,并用 256 kb/s 的速度传送 16 位 48 kHz 取样的立体声声音,其所需的总带宽为 1.5 Mb/s。

在 1.5 Mb/s 的速度下,15 帧/s 的已压缩的视频帧可有 100 千位每帧(100 000 个像素),它所允许的分辨率级别为 400×250 像素每帧。先进的非交错扫描技术使得它运行起来如同以 30 帧/s 的速度进行显示的效果。

新的 MPEG2 目标已将上述速率提高到 4~60 Mb/s。这一改变意味着 MPEG2 可支持更广泛范围的应用,包括数字录像机、视盘播放机、直接广播卫星、CATV 和数字 HDTV。

MPEG 委员会也计划为满足位速为几 kb/s 或更低的需要制定一种新的国际标准,称为 MPEG4。它主要是应用在普通的通信通道中,如公共交换电话网(用调制解调器和 ISDN),低价格的有线及无线网。MPEG4 为低位速的数字编码运动图像和同步音频可以提供各种应用,包括可视电话、电子视频新闻,对视频数据库如 videotex 的远程存取,以及远程传感和监视。MPEG-4 还可以应用某些基于存储的应用,如游戏、电子文档的注释、为听力差的人准备的封闭式字幕及视频邮件。

其他的视频压缩标准还有 Intel 公司的 DVI,它能将动画图像数据以 135 KB/s 的速度传送,声音数据则以 15 KB/s 的速度传送,这使一张 CD-ROM 可存放 75 分钟的动画文件。

9.3.3 多媒体系统的数据和文件格式

多媒体影音文件的格式、标准有很多,以下是用于个人计算机环境的多媒体文件格式,包括:

- ① 多态文本格式(Rich-Text Format, RTF);
- ② 标记图像文件格式(Tagged Image File Format, TIFF);
- ③ 资源文件交换格式(Resource Interchange File Format, RIFF);
- ④ 乐器数字接口(MIDI);
- ⑤ 音频视频交错(AVI)的 Indeo 文件格式。

1. RTF 多态文本格式

多数早期的文本编辑器可以用 ASCII 形式把文本信息保存到文件中,但不保存任何格

式化信息,这就限制了数据交换。因为当文本从一个应用软件移到另一个时,所有格式化信息都丢失了,若要打印,就必须重新输入这些信息。多态文本格式 RTF 就是用于解决这个问题的一种文件格式。多态文本格式不仅保存文本信息本身,也保存文本的格式化信息。只要两个应用软件都含有对格式化信息的解释器,它们之间就可通过将格式化信息翻译为它们自己的格式化控制来共享格式化信息。表 9-6 列出了在 RTF 文档文件中传送的主要格式信息。

RTF 最初的应用目标并不是多媒体系统,但从多媒体系统的角度看,它提供了链接多媒体对象的机制,所以很多应用软件都使用它作为定位、附加、嵌入或连接多媒体对象(如可执行文件、音频文件和视频文件)的手段。

表 9-6 多态文本格式

字符集	字符集确定了支持某一具体实现的字符。字符集组合包括 Windows ANSI、IBM PC、IBM 850 和 Macintosh。例如,n3Mr 字符集引入了一套图形字符用于画方框、边界等等
字体表	字体表列出了文档中用的全部字体。这些字体然后被映射在接收应用软件中可用的字体上,以便显示这些文本
颜色表	颜色表列出了文档中用于加亮文本的颜色(如字符是某一特定颜色而不是黑色)。颜色表也被接收软件映射到最接近它的可用颜色系上,用于显示
文档格式化	RTF 提供了实际的文档页边。文档页边具体规定了段落缩进与文档页边的关系。这一信息帮助确保当由接收应用软件打印文档时,打印页看起来与原始页非常相似,同时段落也显示出相同的相对缩进
节格式化	节中断(和页中断)用于定义分开的段落群。格式化信息具体规定了节上方和下方的空白
段格式化	RTF 说明中定义了控制字符用于具体规定段落调整,制表符位置,左、右和相对于文档页边的第一个段落缩进,以及段落间的空白。段落格式化信息也包括风格样式
通用格式化	这一组中的格式化信息包括诸如脚注、注释、书签和图像这样的项
字符格式化	格式化信息包括黑体、斜体、下划线(连续的、点画的或按字的)、删除线、阴影文本、大纲文本和隐藏文本。它们都用控制字符进行了具体规定。另外下标和上标也由嵌入文本的控制字符说明。注意,字体表中说明的字体和颜色表中说明的颜色都以一种与表有关的方式应用到字符格式中
特殊字符	特殊字符包括连字符、非中断空白、反斜杠等

2. TIFF 文件格式

TIFF 是一种工业标准文件格式,用于表示由扫描仪、视频捕捉卡和平面图形处理应用

软件生成的光栅图像数据。它用不同的数据压缩方式描述了几种彩色空间(或彩色模型)中的光栅图像。由于它是一个带有编译器、处理器和独立于设备的文件格式的操作环境,因而允许在不同的平台上互换文件。

TIFF 文件可由若干个标记组成,这些标记记录了压缩方式、分辨率、每个像素的平面、串长度等状态信息。

TIFF 可以保存的文件包括:

- ① 二值(二进制)图像;
- ② 传真位图;
- ③ 灰度图像;
- ④ 调色板彩色图像;
- ⑤ RGB 全彩色图像;
- ⑥ 拼图和使用 CMYK、YCbCr 和 CIEL * a * 6 彩色模型的图像。

图像压缩方式包括 ITU Group3 2D、ITU Group4 2D、LZW 压缩、JPEG 压缩等。

3. 资源文件交换格式(RIFF)

资源交换文件格式(RIFF)是许多其他文件格式的基础。而且微软也推荐将 RIFF 文件格式结构用于需要新文件格式的应用中。

RIFF 为基于 Windows 的应用程序提供了多媒体文件的框架或包络。它也可以用来将定制文件格式转换为 RIFF 文件格式。这是通过将 RIFF 结构封装在定制文件外面来实现的。例如,通过把 RIFF 信息编码块(“主块”)的结构加在 MIDI 文件上就可将 MIDI 文件格式转换为 RIFF MIDI。

类似于 TIFF, RIFF 是一种标记文件格式,用标记来标识信息。使用标记文件格式,可以更快地进行搜索,因为它允许用 32 位 ASCII 串搜索需要的标记。通过改变标记或加上新的标记可以很容易地处理变化或更新。通过把另一个标记块加到文件上就可为新信息提供更多空间。

每个 RIFF 主块含有主块数据的长度及数据类型。RIFF 规范定义了下列类型的主块:

- ① RIFF 主块 定义了 RIFF 文件的内容。
- ② 表主块 允许嵌入额外的文件信息,如存档位置、版权信息、生成日期等。
- ③ 子主块 当基本主块不够时,允许将更多的信息加到基本主块上。

RIFF 文件格式由称为“主块”的数据块组成。主块与 TIFF 文件的图像文件目录项目类似。RIFF 文件中的第 1 个主块必须是 RIFF 主块,它也许含有一个或多个主块。RIFF 主块数据段中的头 4 个字节分配给格式类型字段,它是一个 ASCII 串 ID(称为标记),用来识别存储在文件中的数据格式:AVI、WAV、RMID 等。表 9-7 列出了 Windows 系统中 RIFF 文件类型使用的扩展文件名。

表 9-7 Windows 多媒体 RIFF 文件类型

文件类型	格式类型	文件扩展名
波形音频文件	WAVE	.WAV
音频视频交错文件	AVI	.AVI
MIDI 文件	RMID	.RMI
独立于装置的位图文件	RDIB	.RDI
调色文件	PAL	.PAL

4. 音频视频交错(AVI)的 RIFF 文件格式

从有声电影一开始创立,语音与画面的同步对电影的成功就非常重要。当演员讲话时,我们看到演员的嘴唇运动并同时听到与嘴唇运动完全同步的语音。不同步的音频和视频令人难以接受,因为人类已养成了看嘴唇运动来完全倾听声音的习惯。

实现数字存储视频剪辑与语音的同步化同样也很重要。AVI 文件格式是按电影模拟设计的。如图 9.17 所示,它含有音频—视频对象的交替帧,以形成交错的文件格式。



图 9.17 AVI 文件的交错音频视频

5. MIDI 文件格式

专业录音师录制音乐时把歌曲的每个成分分别录在不同声道上。声道可以分别进行编辑和处理。在全部声道完全录制和编辑好后,所有声道同时播放,来产生 CD 音质。MIDI 文件格式借用了这种音乐录制的思想来提供存储每一乐器分开的音乐声道的手段,这样它播放时,各部分音乐可以读出并同步化。

类似于 RIFF 文件格式,MIDI 文件格式含有数据主块:头主块和声道主块。

MIDI 规范定义了电子合成器、音序器、节拍器、个人计算机和其他电子乐器的互连性和通信协议。互连性定义了标准接线方式、连接器类型和输入/输出电路,使得这些不同的 MIDI 乐器之间能相互连接。通信协议定义了标准多字节消息。这些消息能控制声音用来发送反应,发送状态和乐器专用的消息。

6. 音频视频交错(AVI)的 Indeo 文件格式

Indeo 视频是由 Intel 体系结构实验室开发的软件压缩标准。它能将未压缩的数字视频文件压缩为 1/5 ~ 1/10。微软公司的 Video for Windows 和苹果(Apple)公司的 QuickTime 中都使用了 Indeo 标准。

Indeo 技术使用了多类“有损”和“无损”压缩技术。Indeo 技术在视频信号被视频捕获卡记录的同时就实时地对它进行压缩。因此未压缩的数据不需存在盘上。从视频摄像机、录像机或激光盘上接收到的以任何标准格式(PAL、NTSC)的视频都可由视频捕获卡转换为数字信号。数字信号要用软件经过下列类型的压缩:

- ① YUV 采样,将像素面积精简压缩到平均彩色值。
- ② 像素差分和时间压缩,通过只存储像素或帧之间的变化信息来压缩数据。
- ③ 游程编码,压缩量化的向量信息。
- ④ 可变内容编码,将可变数量的信息压缩到固定的位数。

压缩的数字化视频文件与音频信息组合成为标准的文件格式(如微软的 AVI 或苹果公司的 QuickTime)进行存储。组合的文件即可被传输和分配用于回放和编辑。

9.3.4 声卡

声卡能带给人们更丰富的信息。对于多媒体微型计算机来说,声卡也是最基本的配置之一。

1. 声卡的基本功能

从总体上讲,声卡的功能可分为这样 3 个部分:模拟音频处理、语音合成、混音和音效处理。

1) 模拟音频处理

声卡可在声音处理软件控制下采样模拟音频信号,再经数字化转成声音文件,并可以回放这些文件或对它们进行编辑等操作。模拟音频信号数字化后占据的磁盘空间很大,以数字化 1 分钟的声音为例,立体声占用的空间为 10 MB。所以声卡在记录和回放数字声音时要进行压缩和解压缩,以节省磁盘空间。声音源可以是话筒、收音机或激光唱盘等。不同声卡和声音处理软件录制的声音文件格式可能不同,但通常它们之间可以相互转换。

2) 语音合成

利用语音合成技术,计算机就能够朗读文本或奏出高保真的合成音乐。目前音乐合成方法有两种,即 FM 合成法和波表合成法。多数声卡用多个人工产生的频率来合成音乐,属于 MIDI 中低档产品。高档声卡使用波表合成法来产生 MIDI 音乐,这种技术在存储器中存放大提琴、鼓、钢琴等乐器的实际波形,再根据 MIDI 文件的内容用这些波形来合成音乐。由于波表非常大,所以声卡上需要大量的存储器来存放它们。

MIDI 音乐文件比普通声音文件所占用的空间小得多,更适于对音乐的进一步处理。通常声卡都有 MIDI 接口,用以和 MIDI 设备相连。声卡可以把来自 MIDI 设备的音乐数据录制成 MIDI 文件进行合成回放,或者利用音乐处理软件进一步处理,也可以把存储在计算机的

中的 MIDI 文件输出到 MIDI 设备中进行外部回放。

3) 混音和音效处理

声卡上都设置有混音器,可以将来自音乐合成器、模拟音频输出和 CD-ROM 驱动器的 CD 模拟音频以不同音量大小混合在一起输出。用声卡唱卡拉 OK 就是利用了声卡上的混音器。少数高档声卡具有音效处理功能,利用数字信号处理技术,通过音效处理芯片,对声音数字信号进行处理,给声音加入混响、延时和回音等效果,使声卡发出的声音更生动。

2. 声卡的基本结构及工作方式

声卡的基本结构如图 9.18 所示,主要包括以下几个部件:

- ① MIDI 输入/输出接口电路;
- ② MIDI 合成器;
- ③ 输入混合电路;
- ④ 模数或数模转换器;
- ⑤ 音频解码芯片;
- ⑥ 语音合成器;
- ⑦ 线路输入/输出接口电路。

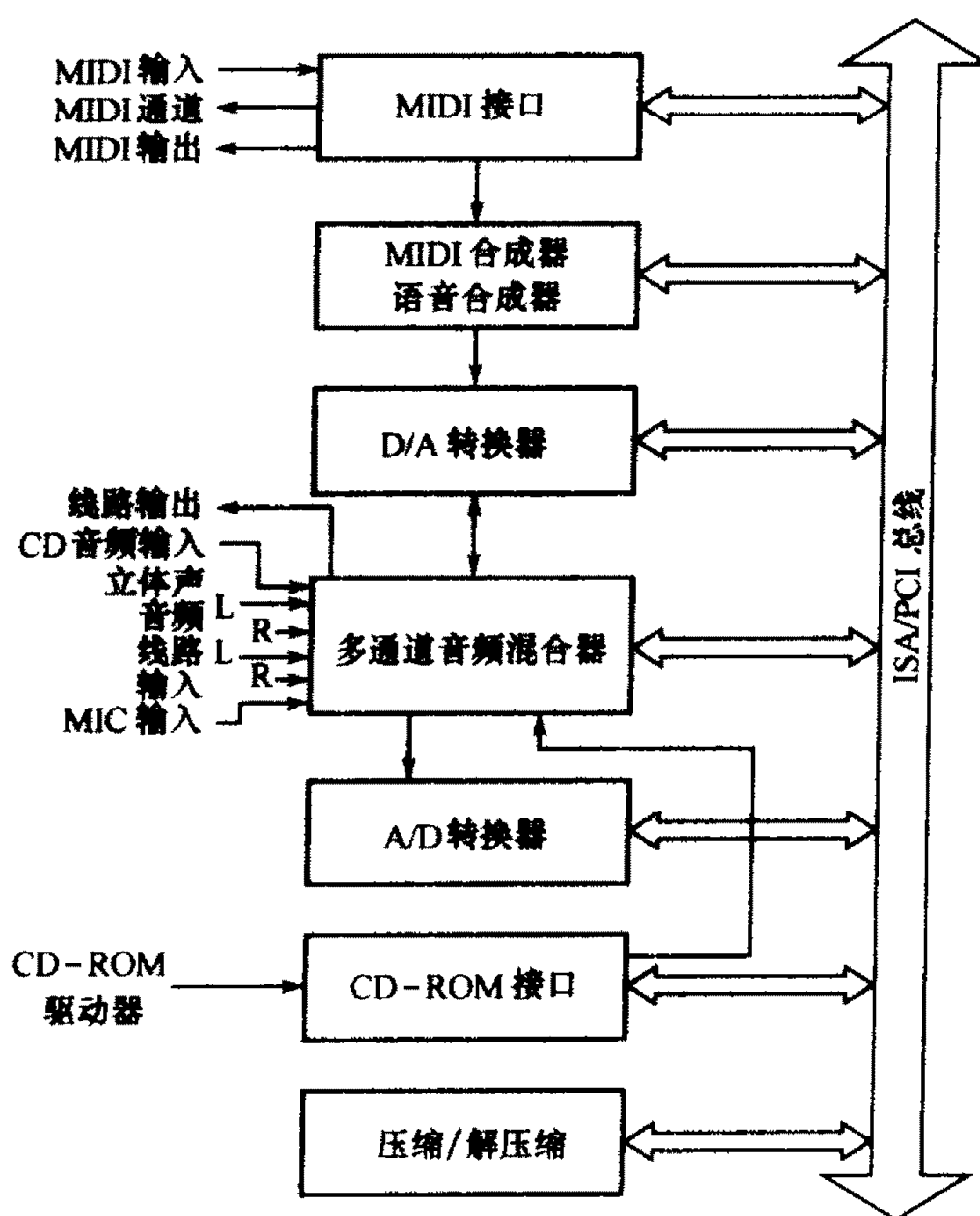


图 9.18 声卡的一般结构

1) MIDI 电路

可以用 FM 合成器或者样本合成器来实现 MIDI 合成器。如果选择 FM 合成器芯片来合成声音,芯片应当支持四种或四种以上运算符。运算符是一些算法上控制的正弦波形,它能支持把这些波形作为载波或调制器相加,这样就生成了声音的复合波形。典型的 FM 合成器芯片可以同时播放 20 个多音质声音(搭配音)。典型的合成器也能同时播放 20 个多音音符。MPC 2.0 规范要求带有 6 个多音音符的 3 种旋律多音质声音和带有 3 个多音音符的 2 种打击多音质声音。

样本合成器是通过存储实际乐器声音的数字样本(即波表)来得到真实乐器声音的。样本存储在 ROM 中或者可从样本库下载入 RAM 区。随着样本记录的改进,样本库对于下载的系统来说也是可以升级的,以便使用新的改进样本,存储的样本在采样频率为 44.1 kHz 的 16 位分辨率情况下以 PCM 格式或 ADCM 格式记录都可。

2) 音频混合器

声卡的音频混合器通常具有用于立体声 CD 音频、立体声音频线路和立体声麦克风 MIC 的外部输入端子。模拟输入经过与 PCM 或 ADPCM 相结合的模拟信号到数字信号的转换来产生数字化的样本。要混合的内部输入是内部 CD 音频、MIDI 合成的声音和 WAVE 格式声音,它们一起被输入混合器。MPC 2.0 规范要求一个麦克风的外部音频输入和 3 个内部输入: CD 音频,数模转换器(DAC)和合成器输入。

3) 模数转换器(ADC)

ADC 从音频混合器那里得到它的输入,并把采集到的模拟信号的幅度转换为 8 位或 16 位二进制数字。

音频工业已经把采样频率标准化为 5.125 kHz(181.41 μ s)、11.025 kHz(90.7 μ s)、22.05 kHz(45.35 μ s)和 44.1 kHz(22.68 μ s)。很明显,越高的采样频率对于保持信号的保真度来说越好。但是,如果信号含有其频率大于采样频率一半的谐波,会导致这些谐波的频率移位,称为产生伪信号。为了防止产生伪信号,声卡中使用低通滤波器来滤掉高于采样频率一半的频率。

分辨率定义为样本的数字值范围。如果采样幅度用 8 位表示,那么幅度的最大级数为 256,范围是 -128 ~ +128。对于 16 位转换器,幅度的最大级数是 65 536,范围是 -32 768 ~ +32 768。

ADC 的另一个重要特性是它的转换速度。为使 ADC 跟上音频信号,转换速度必须快于采样频率。这确保了在把另一个样本交给数字到音频转换器(DAC)来转换之前能正确地把样本转换成它的数字值。

4) 数模转换器(DAC)

DAC 把 WAVE 文件形式的数字输入和 CD 音频转换为模拟信号输出。输出质量取决于

DAC 的分辨率、样本数目和 DAC 的线性度。DAC 输出为每个发送到 DAC 的数字值生成方波。转换过程生成了一系列相互在顶部叠加的方波以产生复合的模拟信号作为幅度包络。低通滤波器过滤掉不需要的谐波来使输出信号变得平滑。

CD 音频用“过采样”技术来提高信号的保真度并减少信号中的噪声。可以用 4X 方式来加入采样信号,就是说,采样电路在每个原始样本之间加入 3 个 0 值样本以形成四样本集。把这些样本输入插值滤波电路中,以计算加在每一样本集上的 3 个 0 值的数值。新样本集中含有原始样本值和 3 个赋予了计算值的样本值。把全部 4 个样本输入以原始信号采样频率的 4X 倍运行的 DAC 中。这就确保了样本值不会由于激光头的机械振动而损失;可以很容易地从剩下的样本中重建平均样本值。

5) 声音压缩和解压缩

多数声音卡都包括用于声音压缩和解压缩的编码解码器。Windows 的 ADPCM 提供了声音压缩的算法。

3. 声卡的安装使用

声卡的安装包括声卡硬件设置、与其他设备的连接以及驱动程序的安装。

1) 声卡硬件设置

硬件设置包括以下几点:

- ① 中断请求 (IRQ);
- ② I/O 地址;
- ③ DMA 或内存基地址;
- ④ 所支持的 CD-ROM 接口类型。

声卡的 I/O 地址设置是为声卡上的寄存器设置端口地址的。在上一节已提到,由于有多个设备要占用中断请求线,声卡所设置的中断请求线不能和其他设备冲突。在为声卡上的寄存器设置端口地址时,也要避免与其他设备冲突,如果声卡以 DMA 方式和内存交换数据,还要设置声卡所要占用的 DMA 通道号,系统的 DMA 通道中有一些已经被其他设备占用,声卡必须使用空闲的 DMA 通道。如果声卡以内存映射方式工作,声卡上所带的存储器会占用系统的地址空间,内存基地址是这一地址空间的开始地址,所以要从相应的内存管理器中去掉这一地址空间,防止同其他设备发生冲突。

有些声卡通过跳线进行硬件设置,有些则是通过软件设置。对软件设置的声卡如果支持即插即用,可在安装驱动程序时由操作系统自动为声卡申请系统资源。

2) 声卡与其他设备的连接

声卡的右端装有与其他设备相连的一些连接头和开关,最常见的有:

- ① MIDI/游戏连接器 用于连接 MIDI 设备或游戏操纵杆;
- ② 线路输出 (Line - Out) 用于连接耳机或外置放大器;

③ 线路输入(Line - In) 用于连接录音机或 CD 音响等设备;

④ 扬声器(Speaker) 用于连接扬声器;

⑤ 麦克风(MIC) 用于连接话筒,输入声音;

3) 设备驱动程序的安装

当完成声卡的硬件设置,并且与其他设备正确连接后,便可以开机安装驱动程序。一般安装程序名为 SETUP 或 INSTALL。安装程序会自动安装声卡的驱动程序,并根据用户的选择进行应用程序的安装。在安装过程中,可能会要求用户输入一些配置信息,注意要与硬件参数统一。有些声卡不提供安装程序,这时应打开控制面板,进入设备管理器用手工进行安装。

9.3.5 视频获取卡

视频获取卡也称视频捕捉卡,是计算机系统中的一块电路板,用于把各种不同的音频和视频输入信号转换为数字信号。它是多媒体制作系统中的一种重要输入手段。视频获取卡支持各种电视信号制式,如亚洲的 PAL、北美洲的 NTSC、欧洲及中东地区的 SECAM 等。视频获取卡的输入源可以是视频摄像机、录像机或视频网络上的视频图像,其主要由以下几个部件组成:

- ① 视频 S - Video 输入接口;
- ② 视频 A/D(模数转换器);
- ③ 带有算术逻辑单元的输入查找表,用来对图像进行数学和逻辑操作;
- ④ 图像帧缓冲器用来存储要进行图像处理和显示的图像;
- ⑤ 用来压缩和解压缩图像数据的压缩和解压缩电路;
- ⑥ 音频压缩解压缩处理器;
- ⑦ 含有灰度值和彩色值的输出彩色查找表;
- ⑧ 视频 D/A(数模转换器);
- ⑨ 立体声音频线路输入、CD 音频输入和 MIC 输入的接口。

视频获取卡的体系结构如图 9.19 所示,下面简要介绍其中的各个部分。

1) 视频通道多路转接器

视频通道多路转接器用于转接不同视频标准的多个输入:

- ① 60 Hz 电视信号的 RS 170/RS 170A 标准或 NTSC 标准;
- ② 50 Hz 电视信号的 PAL 或 CCIR 标准;
- ③ 25 Hz 电视信号的 SECAM 标准和质量增强的视频信号的 S - Video 输入。

视频通道多路转接器允许在程序控制下,选择视频通道并转换相应的控制电路,转换的电路还包括水平和垂直同步电路及相关的放大器,以及图像输出的视频混合器。

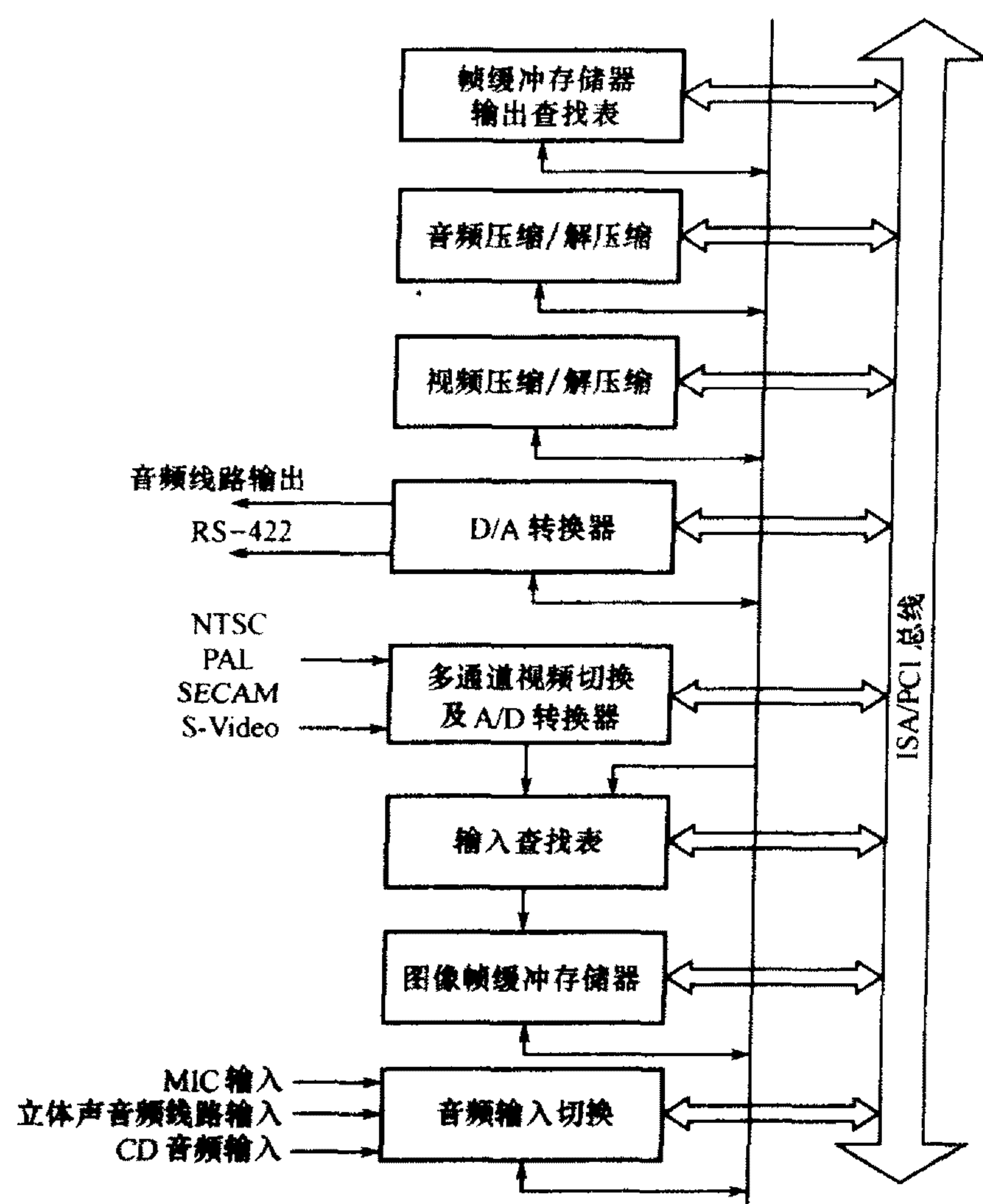


图 9.19 视频获取卡的一般结构

2) 模数转换器(ADC)

ADC 从视频多路转接器中获取输入信号,并把采集的模拟信号幅度转换为单色的 8 位数字值或彩色的 24 位数字值。像素的分辨率是由用来表示像素值的位数确定的,8 位能产生 256 级的灰度色调或 256 种彩色;24 位用来生成具有 1670 万种彩色范围的彩色像素。

3) 输入查找表

输入查找表及算术逻辑单元(ALU)允许在像素基础上和图像帧基础上完成图像处理操作。像素图像处理操作是用于调节图像亮度和对比度的直方图扩展或直方图收缩,以及使图像变亮或变暗的直方图滑动完成的。基于帧的图像处理操作完成的是逻辑和算术操作。使用查找表的主要优点是加快了图像处理的速度。

4) 图像帧缓冲存储器

1 024 × 1 024 × 24 的图像帧缓冲存储器来存储用于处理和显示的图像。帧缓存器具有

两个端口：一个端口由输入查找表使用，查找表的内容存储在帧缓存器中；另一个端口用于主机 CPU，主机 CPU 可以读帧缓存器中的内容，并在必要时对它进行操作。

5) 视频压缩/解压缩

视频压缩/解压缩处理器是用来压缩和解压缩静止图像数据和视频数据的。尽管视频工业是在实时视频的 MPEG-2 标准上建立起来的，但仍有其他的一些标准在使用之中，如运动 JPEG 和 CCITT H.261。为了在一块卡上为压缩和解压缩提供多个标准算法，视频获取卡需要有可编程的视频压缩/解压缩器(编码解码器)。“可编程”为获取卡提供了灵活性，当算法改进时，可以生成新的微程序。

6) 音频压缩

对声音按 MPEG-2 音频标准进行编码压缩，生成数据化的声音。

7) 帧缓冲存储器输出查找表

帧缓存器中存放的像素数据作为输出查找表的索引。输出查找表生成单色的 8 位像素值或彩色的 24 位像素值。查找表的内容由应用程序定义，并由主 CPU 编程来表示单色的灰度值范围或彩色输出的彩色值范围。例如，8 位值表示了指定像素的灰度值，或 24 位值表示了指定像素的彩色值。输出查找表的输出信息送到数模转换器(DAC)中即可把像素转换成模拟信号。

8) 模拟输出混合器

把来自 SVGA D/A 转换器的输出与来自图像帧缓冲存储器 D/A 转换器的输出相混合以生成叠加的输出信号。涉及的基本部件包括显示图像帧缓存器和 SVGA 显示缓冲存储器。把 SVGA 缓存器的内容非破坏性地叠加在图像帧缓存器或现场视频上，就可以在 SVGA 显示器上显示现场视频图像。

习 题 九

- 9.1 在微型计算机系统基本配置中，常用的输入设备有哪些？输出设备有哪些？
- 9.2 键盘接口控制器的主要功能是什么？
- 9.3 键盘与 CPU 之间以何种方式通信？
- 9.4 试简述键盘缓冲区的作用。
- 9.5 按照不同的工作原理，鼠标可分为哪几种？它们各自具有什么样的特点？
- 9.6 显示器有哪两种显示方法？
- 9.7 显示接口卡上主要包括哪几个部件？各部件的主要功能是什么？
- 9.8 简述并行打印机的工作原理。
- 9.9 要使一台计算机与局域网相连，需要在系统中加入什么设备？如果是通过电话线与广域网连接

呢?

- 9.10 选用网卡需要考虑哪几方面的因素?
- 9.11 什么是设备驱动程序? 其主要作用是什么?
- 9.12 DOS 设备驱动程序主要由哪三个部分组成?
- 9.13 何为即插即用? 要做到即插即用, 要求所安装的硬件要具备什么条件?
- 9.14 何为多媒体计算机? 它与一般的商用计算机有什么不同?
- 9.15 什么是 MIDI 设备? 什么是 MIDI 文件?
- 9.16 为什么要对多媒体数据进行压缩处理? 其压缩标准包括哪两种? 它们各自具有什么特点?
- 9.17 MPEG 和 JPEG 两种压缩方式有哪些相同点? 有哪些不同点?
- 9.18 多媒体个人计算机包括哪几种影音文件格式? 它们各自的主要用途是什么?
- 9.19 声卡是根据什么分类的? 常用有哪几种?
- 9.20 视频采集卡的用途是什么? 它由哪几个部分组成?

附录

附录 A ASCII 码表

列		0	1	2	3	4	5	6	7
行	高位	000	001	010	011	100	101	110	111
	低位								
0	0000	NUL	DLE	SP	0	@	P	,	p
L	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	\$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(8	H	X	h	x
9	1001	HT	EM)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[k	
C	1100	FF	FS	,	<	L	\	l	
D	1101	CR	GS	-	=	M]	m	
E	1110	SO	RS	.	>	N	Ω	n	~
F	1111	SI	US	/	?	O	-	o	DEL

注：表中的 00H ~ 1FH 以及 7FH 为控制符，不可显示；其余的为可显示字符。

ASCII 码表中控制符号的定义

NUL	Null	空白	DLE	Data Link Escape	转义
SOH	Start Of Heading	标题开始	DC1	Device Control 1	设备控制 1
STX	Start Of Text	正文开始	DC2	Device Control 2	设备控制 2
ETX	End Of Text	正文结束	DC3	Device Control 3	设备控制 3
EOT	End Of Transmit	传输结束	DC4	Device Control 4	设备控制 4
ENQ	Enquiry	询问	NAK	Negative Acknowledge	否定
ACK	Acknowledge	承认	SYN	Synchronize	同步
BEL	Bell	响铃	ETB	End of Transmitted Block	信息组结束
BS	Backspace	退格	CAN	Cancel	作废
HT	Horizontal Tab	横向制表	EM	End of Medium	纸尽
LF	Line Feed	换行	SUB	Substitute	取代
VT	Vertical Tab	纵向制表	ESC	Escape	换码
FF	Form Feed	换页	FS	File Separator	文件分隔符
CR	Carriage Return	回车	GS	Group Separator	组分隔符
SO	Shift Out	移出	RS	Record Separator	记录分隔符
SI	Shift In	移入	US	Unit Separator	单元分隔符
SP	Space	空格	DEL	Delete	删除

附录 B 8086/8088 指令简表

汇编格式	指令的操作
1. 数据传送指令	
MOV <i>dest, source</i>	数据传送。数据从 <i>source</i> 到 <i>dest</i>
CBW	将累加器 AL 中的字节扩展到 AX
CWD	将 AX 中的字操作数扩展到 DXAX
LAHF	FLAGS 低 8 位装入 AH 寄存器
SAHF	AH 寄存器内容送到 FLAGS 低 8 位
LDS <i>dest, source</i>	设定数据段指针
LES <i>dest, source</i>	设定附加段指针
LEA <i>dest, source</i>	将 <i>source</i> 的有效地址装入 <i>dest</i>
PUSH <i>source</i>	将一个字压入栈顶
POP <i>dest</i>	将一个字从栈顶弹出

续表

汇编格式	指令的操作
PUSHF	将标志寄存器 FLAGS 的内容压入栈顶
POPF	将栈顶内容弹出到标志寄存器 FLAGS
XCHG <i>dest, source</i>	<i>source</i> 和 <i>dest</i> 的内容交换
XLAT <i>source</i>	表转换
2. 算术运算指令	
A AA	加法的 ASCII 调整
A AD	除法的 ASCII 调整
AAM	乘法的 ASCII 调整
AAS	减法的 ASCII 调整
DAA	加法的十进制调整
DAS	减法的十进制调整
MUL <i>source</i>	无符号乘法。累加器 \times <i>source</i> 累加器或 DX
IMUL <i>source</i>	有符号数乘法
DIV <i>source</i>	无符号除法
IDIV <i>source</i>	有符号数除法
ADD <i>dest, source</i>	加法 $Dest + source$ Dest
ADC <i>dest, source</i>	带进位加 $Dest + source + CF$ Dest
SUB <i>dest, source</i>	减法 $Dest - source$ Dest
SBB <i>dest, source</i>	带借位减 $Dest + source - CF$ Dest
CMP <i>dest, source</i>	比较 $Dest - source$
INC <i>dest</i>	加 1 $Dest + 1$ Dest
DEC <i>dest</i>	减 1 $Dest - 1$ Dest
NEG <i>dest</i>	求补 $0 - Dest$ Dest
3. 逻辑运算指令	
AND <i>dest, source</i>	逻辑“与” $Dest \wedge source$ Dest
OR <i>dest, source</i>	逻辑“或” $Dest \vee source$ Dest
XOR <i>dest, source</i>	逻辑“异或” $Dest \vee source$ Dest
NOT <i>dest</i>	逻辑“非” Dest Dest
TEST <i>dest, source</i>	测试 $Dest \wedge source$
4. 移位指令	
RCL <i>dest, count</i>	带进位位循环左移 <i>count</i> 位
RCR <i>dest, count</i>	带进位位循环右移 <i>count</i> 位
ROL <i>dest, count</i>	不带进位位循环左移 <i>count</i> 位

续表

汇编格式	指令的操作
ROR <i>dest, count</i>	不带进位位循环右移 <i>count</i> 位
SHL/SAL <i>dest, count</i>	逻辑左移/算术左移 <i>count</i> 位
SHR <i>dest, count</i>	逻辑右移 <i>count</i> 位
SAR <i>dest, count</i>	算术右移 <i>count</i> 位
5. 串操作指令	
MOVS/MOVS _B /MOVSW <i>dest, source</i>	字符串从 <i>source</i> 传送到 <i>dest</i>
CMPS/CMPS _B /CMPSW <i>dest, source</i>	<i>Source</i> 中字符串与 <i>dest</i> 中字符串比较
LODS/LODS _B /LODSW <i>source</i>	装入字节串或字串到累加器
STOS/STOS _B /STOSW <i>dest</i>	存储字节串或字串
SCAS/SCAS _B /SCASW <i>dest</i>	字符串扫描
6. 程序控制指令	
CALL <i>dest</i>	调用一个过程(子程序)
RET [弹出字节数(必须为偶数)]	从过程(子程序)返回
INT <i>int_type</i>	软件中断
INTO	溢出中断
IRET	从中断返回
JMP <i>dest</i>	无条件转移
JG/JNLE <i>short_label</i>	大于或不小于等于转移
JGE/JNL <i>short_label</i>	大于等于或不小于转移
JL/JNGE <i>short_label</i>	小于或不大于等于转移
JLE/JNG <i>short_label</i>	小于等于或不大于转移
JA/JNBE <i>short_label</i>	高于或不低于等于转移
JAE/JNB <i>short_label</i>	高于等于或不低于转移
JB/JNAE <i>short_label</i>	低于或不高于等于转移
JBE/JNA <i>short_label</i>	低于等于或不高于转移
JO <i>short_label</i>	溢出标志为 1 转移(溢出转移)
JNO <i>short_label</i>	溢出标志为 0 转移(无溢出转移)
JS <i>short_label</i>	符号标志为 1 转移(结果为负转移)
JNS <i>short_label</i>	符号标志为 0 转移(结果为正转移)
JNO <i>short_label</i>	溢出标志为 0 转移(无溢出转移)
JC <i>short_label</i>	进位标志为 1 转移(有进位转移)
JNC <i>short_label</i>	进位标志为 0 转移(无进位转移)
JZ/JE <i>short_label</i>	零标志为 1 转移(等于或为 0 转移)

续表

汇编格式	指令的操作
JNZ/JNE <i>short_label</i>	零标志为 0 转移(不等于或不为 0 转移)
JP/JPE <i>short_label</i>	奇偶标志为 1 转移(结果中有偶数个 1 转移)
JNP/JPO <i>short_label</i>	奇偶标志为 0 转移(结果中有奇数个 1 转移)
JCXZ <i>short_label</i>	若 CX = 0 则转移
LOOP <i>short_label</i>	CX ≠ 0 时循环
LOOPE/LOOPZ <i>short_label</i>	CX ≠ 0 且 ZF = 1 时循环
LOOPNE/LOOPNZ <i>short_label</i>	CX ≠ 0 且 ZF = 0 时循环
STC	进位标志置 1
CLC	进位标志置 0
CMC	进位标志取反
STD	方向标志置 1
CLD	方向标志置 0
STI	中断标志置 1(允许可屏蔽中断)
CLI	中断标志置 0(禁止可屏蔽中断)
ESC	CPU 交权
HLT	停机
LOCK	总线封锁
NOP	无操作
WAIT	等待至 $\overline{\text{TEST}}$ 信号有效为止
7. 输入输出指令	
IN <i>acc, port</i>	从端口 <i>port</i> 输入字节或字到累加器 AL 或 AX
OUT <i>port, acc</i>	将累加器 AL 或 AX 中的字节或字输出到端口 <i>port</i>

注：

- dest*：目的操作数,目的串
- source*：源操作数,源串
- acc*：累加器
- count*：计数值
- int_type*：中断类型号
- short_label*：短距离标号
- port*：外设端口地址

附录 C 8086/8088 的中断系统

C.1 中断类型分配

类 别	中断类型码 (Hex)	功 能
软件 自陷 和 NMI 中断	0	除法错
	1	单步
	2	NMI 中断
	3	断点
	4	溢出
	5	屏幕拷贝
	6,7	未使用
主 8259 管理的中断 (可屏蔽中断)	8	系统定时/计数器
	9	键盘
	A	未使用(从 8259A 与此中断级连)
	B	COM2
	C	COM1
	D	并口 2(打印机)
	E	软盘驱动器
ROM - BIOS 软中断	F	并口 1(打印机)
	10	屏幕显示
	11	检测系统配置
	12	检测存储器容量
	13	磁盘 I/O
	14	异步通信 I/O
	15	盒式磁带机, I/O 系统扩展
	16	键盘 I/O
	17	打印机 I/O
	18	ROM - BASIC 入口
	19	系统自举(冷起动)
	1A	日时钟 I/O
供用户 链接的中断	1B	键盘 Ctrl - Break 中断
	1C	定时/计数器产生的中断(每 55 ms 产生一次)
数据表 指针	1D	显示器初始化参数
	1E	软盘参数
	1F	显示图形字符

续表

类 别	中断类型码(Hex)	功 能
DOS 软中断	20	程序正常结束
	21	系统功能调用
	22	程序结束退出
	23	Ctrl - Break 退出
	24	严重错误处理
	25	绝对磁盘读功能
	26	绝对磁盘写功能
	27	程序驻留并退出
	28 ~ 2E	DOS 保留
	2F	假脱机打印
	30 ~ 3F	DOS 保留
杂类	40	软盘 I/O 重定向
	41	硬盘参数
	42 ~ 5F	系统保留
	60 ~ 6F	保留给用户使用
从 8259 管理的中断 (可屏蔽中断)	70	实时时钟
	71	IRQ9(INT 0AH 重定向)
	72	IRQ10(保留)
	73	IRQ11(保留)
	74	IRQ12(保留)
	75	协处理器
	76	硬盘控制器
	77	IRQ15(保留)
其他	78 ~ 7F	未使用
	80 ~ F0	BASIC 占用
	F1 ~ FF	未使用

C.2 DOS 系统功能调用简表

功能号	功能	入口参数	出口参数
1. 设备管理功能			
01H	键盘输入		AL = 输入字符
02H	显示器输出	DL = 输出字符	
03H	串行设备输入字符		AL = 输入字符
04H	串行设备输出字符	DL = 输出字符	
05H	打印机输出	DL = 输出字符	
06H	直接控制台 I/O	DL = FFH(输入) DL = 输出字符(输出)	AL = 输入字符

续表

功能号	功能	入口参数	出口参数
07H	直接控制台输入(无回显)		AL = 输入字符
08H	键盘输入(无回显)		AL = 输入字符
09H	显示字符串	DS:DX = 字符缓冲区首址	
0AH	带缓冲的键盘输入(字符串)	DS:DX = 键盘缓冲区首址	
0BH	检查标准输入状态		AL = 0 无键入 AL = FFH 有键入
0CH	清除键盘缓冲区,然后输入	AL = 功能号(1,6,7,8,A)	(与指定的功能相同)
0DH	刷新 DOS 磁盘缓冲区		
0EH	选择磁盘	DL = 盘号	AL = 系统中盘的数目
19H	取当前盘盘号		AL = 盘号
1AH	设置磁盘传送缓冲区(DTA)	DS:DX = DTA 首址	
1BH	取当前盘文件分配表(FAT)信息		DS:BX = 盘类型字节地址 DX = FAT 表项数 AL = 每簇扇区数 CX = 每扇区字节数
1CH	取指定盘文件分配表(FAT)信息	DL = 盘号	(同上)
2EH	置写校验状态	DL = 0, AL = 状态(0 关, 1 开)	AL = 0 成功, AL = FFH 失败
54H	取写校验状态		AL = 状态(0 关, 1 开)
36H	取盘剩余空间	DL = 盘号	BX = 可用簇数 DX = 总簇数 AX = 每簇扇区数 CX = 每扇区字节数
2FH	取磁盘传送缓冲区(DTA)首址		ES:BX = DTA 首址
2. 文件管理功能			
29H	建立文件控制块 FCB	DS:SI = 文件名字符串首址 ES:DI = FCB 首址 AL = 0EH 非法字符检查	ES:DI = 格式化后的 FCB 首址 AL = 0 标准文件 AL = 1 多义文件 AL = FFH 非法盘符
16H	建立文件(FCB 方式)	DS:DX = FCB 首址	AL = 0 成功 AL = FFH 目录区满
0FH	打开文件(FCB 方式)	DS:DX = FCB 首址	AL = 0 成功 AL = FFH 未找到
10H	关闭文件(FCB 方式)	DS:DX = FCB 首址	AL = 0 成功 AL = FFH 已换盘
13H	删除文件(FCB 方式)	DS:DX = FCB 首址	AL = 0 成功 AL = FFH 未找到
14H	顺序读一个记录	DS:DX = FCB 首址	AL = 0 成功 AL = 1 文件结束 AL = 3 缓冲不满
15H	顺序写一个记录	DS:DX = FCB 首址	AL = 0 成功 AL = FFH 盘满

续表

功能号	功能	入口参数	出口参数
21H	随机读一个记录	DS:DX = FCB 首址	AL = 0 成功 AL = 1 文件结束 AL = 3 缓冲不满
22H	随机写一个记录	DS:DX = FCB 首址	AL = 0 成功 AL = FFH 盘满
27H	随机读多个记录	DS:DX = FCB 首址 CX = 记录数	AL = 0 成功 AL = 1 文件结束 AL = 3 缓冲不满
28H	随机写多个记录	DS:DX = FCB 首址 CX = 记录数	AL = 0 成功 AL = FFH 盘满
24H	置随机记录号	DS:DX = FCB 首址	
3CH	建立文件(文件号方式)	DS:DX = 文件名首址 CX = 文件属性	若 CF = 0, AX = 文件号 否则失败, AX = 错误代码
3DH	打开文件(文件号方式)	DS:DX = 文件名首址 AL = 0 只读 AL = 1 只写 AL = 2 读/写	若 CF = 0, AX = 文件号 否则失败, AX = 错误代码
3EH	关闭文件(文件号方式)	BX = 文件号	CF = 0 成功, 否则失败
41H	删除文件	DS:DX = 文件名首址	若 CF = 0, 成功 否则失败, AX = 错误代码
3FH	读文件(文件号方式)	BX = 文件号 CX = 读的字节数 DS:DX = 缓冲区首址	AX = 实际读的字节数
40H	写文件(文件号方式)	BX = 文件号 CX = 写的字节数 DS:DX = 缓冲区首址	AX = 实际写的字节数
42H	移动文件读写指针	BX = 文件号 CX:DX = 位移量 AL = 0 从文件头开始移动 AL = 1 从当前位置移动 AL = 2 从文件尾倒移	若 CF = 0, 成功 DX:AX = 新的指针位置 否则失败 AX = 1 无效的移动方法 AX = 6 无效的文件号
45H	复制文件号	BX = 文件号 1	若 CF = 0, AX = 文件号 2 否则失败, AX = 错误代码
46H	强制复制文件号	BX = 文件号 1 CX = 文件号 2	若 CF = 0, CX = 文件号 1 否则失败, AX = 错误代码
4BH	装入一个程序	DS:DX = 程序路径名首址 ES:BX = 参数区首址 AL = 0 装入后执行 AL = 3 仅装入	若 CF = 0, 成功 否则失败

续表

功能号	功能	入口参数	出口参数
44H	设备文件 I/O 控制	BX = 文件号 AL = 0 取状态 AL = 1 置状态 DX AL = 2 读数据 * AL = 3 写数据 * AL = 6 取输入状态 AL = 7 取输出状态 (* DS:DX = 缓冲区首址, CX = 读写的字节数)	DX = 状态
3. 目录操作功能			
11H	查找第一个匹配文件(FCB 方式)	DS:DX = FCB 首址	AL = 0 成功 AL = FFH 未找到
12H	查找下一个匹配文件(FCB 方式)	DS:DX = FCB 首址	AL = 0 成功 AL = FFH 未找到
23H	取文件长度(结果在 FCB RR 中)	DS:DX = FCB 首址	AL = 0 成功 AL = FFH 失败
17H	更改文件名(FCB 方式)	DS:DX = FCB 首址 (DS:DX + 17) = 新文件名	AL = 0 成功 AL = FFH 失败
4EH	查找第一个匹配文件	DS:DX = 文件路径名首址 CX = 文件属性	若 CF = 0 成功 DTA 中有该文件的信息 否则失败, AX = 错误代码
4FH	查找下一个匹配文件	DTA 中有 4EH 得到的信息	(同 4EH)
43H	置/取文件属性	DS:DX = 文件名首址 AL = 0 取文件属性 AL = 1 置文件属性(CX)	若 CF = 0 成功 CX = 文件属性(读时) 否则失败, AX = 错误代码
57H	置/取文件日期和时间	BX = 文件号 AL = 0 取日期时间 AL = 1 置日期时间(DX: CX)	若 CF = 0 成功 DX: CX = 日期和时间 否则失败, AX = 错误代码
56H	更改文件名	DS:DX = 老文件名首址 ES:DI = 新文件名首址	
39H	建立一个子目录	DS:DX = 目录路径串首址	若 CF = 0 成功, 否则失败
3AH	删除一个子目录	DS:DX = 目录路径串首址	若 CF = 0 成功, 否则失败
3BH	改变当前目录	DS:DX = 目录路径串首址	若 CF = 0 成功, 否则失败
47H	取当前目录路径名	DL = 盘号 DS:SI = 字符串首址	若 CF = 0 成功 DS:SI = 目录路径名首址 否则失败, AX = 错误代码
4. 其他功能			
00H	程序结束, 返回操作系统		
31H	终止程序并驻留在内存	AL = 退出码 DX = 程序长度	

续表

功能号	功能	入口参数	出口参数
4CH	终止当前程序, 返回调用程序	AL = 退出码	
4DH	取退出码		AL = 退出码
33H	置取 Ctrl - Break 检查状态	AL = 0 取状态 AL = 1 置状态 (DL = 0 关, DL = 1 开)	DL = 状态 (AL = 0 时)
25H	置中断向量	AL = 中断类型号 DS:DX = 中断服务程序入口	
35H	取中断向量	AL = 中断类型号	ES:BX = 中断服务程序入口
26H	建立一个程序段	DX = 段号	
48H	分配内存空间	BX = 申请内存数量 (以 16B 为单位)	CF = 0 成功 AX:0 = 分配内存首址 否则失败 BX = 最大可用内存空间
49H	释放内存空间	ES:0 = 释放内存块的首址	CF = 0 成功 否则失败, AX = 错误代码
4AH	修改已分配的内存空间	ES = 已分配的内存段地址 BX = 新申请的数量	CF = 0 成功 AX:0 = 分配内存首址 否则失败 BX = 最大可用内存空间
2AH	取日期		CX:DX = 日期
2BH	置日期	CX:DX = 日期	AL = 0 成功 AL = FFH 失败
2CH	取时间		CX:DX = 时间
2DH	置时间	CX:DX = 时间	AL = 0 成功 AL = FFH 失败
30H	取 DOS 版本号		AL = 版本号, AH = 发行号
38H	取国家信息	DS:DX = 信息存放地址 AL = 0	CF = 0 成功 DS:DX = 信息区地址

附录 D 常用伪指令简表

伪指令名	伪指令操作码及格式	指令的操作
数据定义伪指令	变量名 DB 操作数[,操作数,操作数...]	定义字节变量
	变量名 DW 操作数[,操作数,操作数...]	定义字变量
	变量名 DD 操作数[,操作数,操作数...]	定义双字变量
	变量名 DQ 操作数[,操作数,操作数...]	定义 8B 变量
	变量名 DT 操作数[,操作数,操作数...]	定义 10B 变量
重复操作符	[变量名] 数据定义伪操作 n DUP (?)	定义初值为随机数的 n 个元素大小的数据区
段定义伪指令	段名 SEGMENT : 段名 ENDS	定义一个逻辑段
设定段寄存器伪指令	ASSUME 段寄存器名: 段名 [,段寄存器名: 段名,...]	说明所定义的逻辑段的属性
符号定义伪指令	符号名 EQU 表达式	将表达式的值赋予符号名
结束伪指令	END [标号]	位于源程序的最后,表示源程序一个模块的结束
过程定义伪指令	过程名 PROC [NEAR/FAR] : RET 过程名 ENDP	定义一个过程
宏命令伪指令	宏命令名 MACRO [形式参数,...] (宏定义体) ENDM	定义宏

参考文献

- 1 Predko M 著. PC 机接口内幕. 陈逸译. 北京: 中国电力出版社, 2002
- 2 冯博琴主编. 微型计算机原理与接口技术. 北京: 清华大学出版社, 2002
- 3 Barry B 著. Intel 微处理器全系列: 结构、编程与接口(第五版). 金惠华等译. 北京: 电子工业出版社, 2001
- 4 王闵编著. 计算机组成原理. 北京: 电子工业出版社, 2001
- 5 薛均义, 姚燕南编. 微型计算机原理. 西安: 西安电子科技大学出版社, 2000
- 6 冯博琴主编. 硬件技术基础. 北京: 人民邮电出版社, 2000
- 7 李伯成等编. 微型计算机原理及应用. 西安: 西安电子科技大学出版社, 1999
- 8 郑纬民, 汤志忠编. 计算机系统结构. 北京: 清华大学出版社, 1998
- 9 王永山等编. 微型计算机原理与应用. 西安: 西安电子科技大学出版社, 1999
- 10 Barry B 著. Programming the 80286, 80386, 80486, and Pentium - Based Personal Computer. Prentice Hall, 1996
- 11 Intel Corporation. The 8086 Family User's Manual, 1979
- 12 俸远祯等编. 计算机组成原理与汇编语言程序设计. 北京: 电子工业出版社, 1997

郑 重 声 明

高等教育出版社依法对本书享有专有出版权。任何未经许可的复制、销售行为均违反《中华人民共和国著作权法》,其行为人将承担相应的民事责任和行政责任,构成犯罪的,将被依法追究刑事责任。为了维护市场秩序,保护读者的合法权益,避免读者误用盗版书造成不良后果,我社将配合行政执法部门和司法机关对违法犯罪的单位和个人给予严厉打击。社会各界人士如发现上述侵权行为,希望及时举报,本社将奖励举报有功人员。

反盗版举报电话: (010) 58581897/58581698/58581879/58581877

传 真: (010) 82086060

E - mail: dd@hep.com.cn 或 chenrong@hep.com.cn

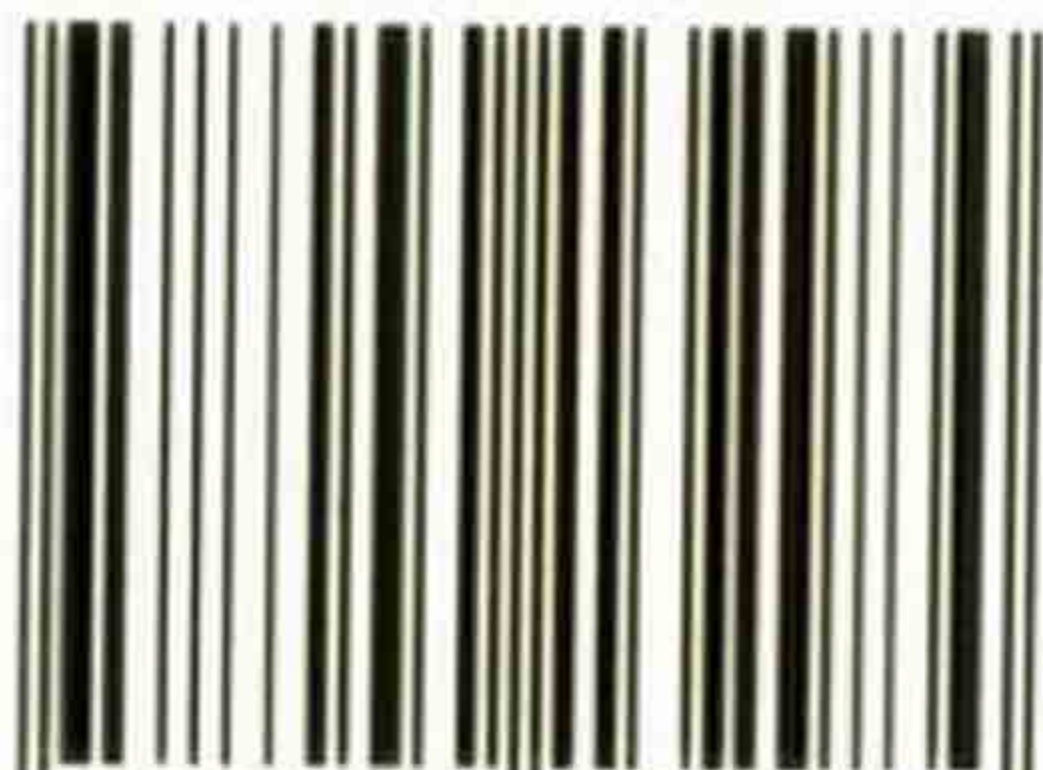
通信地址: 北京市西城区德外大街 4 号

高等教育出版社法律事务部

邮 编: 100011

购书请拨打电话: (010)64014089 64054601 64054588

ISBN 7-04-013298-2



9 787040 132984 >

定价：35.20 元